UNIVERSIDAD CARLOS III DE MADRID
DEPARTAMENTO DE INGENIERÍA TELEMÁTICA

IMPLEMENTACIÓN DE UN TUTOR INTELIGENTE PARA LA AYUDA ANTE ERRORES DE COMPILACIÓN BASADO EN ONTOLOGÍAS Y TÉCNICAS DE WEB SEMÁNTICA

PROYECTO FIN DE CARRERA:

INGENIERÍA TÉCNICA DE TELECOMUNICACIONES:
SONIDO E IMAGEN

- Autor: **Belén del Campo Romero**
- Tutor: **Pedro José Muñoz Merino**

Leganés, Noviembre de 2013

Agradecimientos

Quisiera aprovechar estas líneas para dar las gracias a todas esas personas que han hecho posible la finalización de mi carrera:

A mi tutor, Pedro Muñoz Merino, que me ha brindado la oportunidad de realizar este proyecto y aprender de él. Así como a Aria Estevez Ayres y a Abelardo Pardo que también aportaron su granito de arena. Sin vuestro trabajo y dedicación, no hubiera sido posible.

Y en general a todos mis profesores, desde el colegio hasta la universidad, por todo lo que he aprendido gracias a vosotros.

A mis padres, vuestros valores, consejos y comprensión, me han servido de guía para alcanzar mis metas. Gracias por creer en mí.

A mi sobrina, me esforzaré para que estés orgullosa de tu 'tata' y veas en mí un ejemplo a seguir. Y por supuesto al resto de mi familia, sois un pilar fundamental en el que apoyarme.

A mis amigos de siempre, que siempre están ahí.

A mis compañeros de la universidad, por no dejarme caer cuando el final se veía tan lejano, animarme en los momentos duros y reír en los momentos buenos. Vuestro apoyo durante toda la carrera ha sido muy importante para mí, y lo mejor es que os habéis convertido en buenos amigos y sé que podré seguir contando con vosotros.

A mis compañeros de trabajo, por su esfuerzo y cariño. Gracias por compartir vuestra experiencia y conocimiento conmigo. He aprendido mucho a vuestro lado, tanto profesional como personalmente.

¡Muchas gracias a todos!

Resumen

Los estudiantes que comienzan a programar en C suelen cometer errores de compilación, cuya interpretación y resolución suele necesitar revisión del material del curso o búsqueda de información específica, ya que solo con los mensajes de error del compilador no es siempre fácil determinar la causa.

En este proyecto se ha desarrollado un tutor inteligente que, a partir de los errores de compilación obtenidos, entra en un diálogo con el alumno y genera recomendaciones con enlaces al material del curso que ayudan en el aprendizaje del alumno, evitando así una búsqueda manual. La aplicación descrita se ha integrado en una máquina virtual que utilizan los alumnos de la asignatura 'Arquitectura de Sistemas' donde se aprende a programar en C.

Para la creación de este tutor inteligente se han utilizado técnicas de Web Semántica y se han desarrollado ontologías sobre lo errores más comunes que los alumnos cometen mientras programan con todas sus posibles causas y una ontología sobre el propio dominio de programación en C. Los diferentes recursos educativos se han anotado según esas ontologías utilizando RDF. Además, se utiliza el entorno Jena y la base de datos SDB para desarrollar la aplicación.

Palabras claves: Web Semántica, ontología, freedback, recomendador, RDF, SPARQL y OWL.

Índice general

Índice de figuras

1. INTRODUCCIÓN

1.1. Motivación

Las ideas iniciales que motivaron este proyecto, se recogen en un trabajo anterior [1]. En él se presenta un marco ontológico para la generación de realimentación adaptativa con el fin de ofrecer una respuesta eficaz orientada a estudiantes que aprenden a programar. Para ello, se explica la necesidad de las siguientes ontologías: una del propio dominio de programación, una con los posibles tipos de error que el alumno puede cometer y una con las causas de dichos errores. Todas ellas conectadas por otra ontología intermediaria. Además, la realimentación adaptativa depende del perfil del estudiante, que debe ser obtenido de interacciones anteriores con el apoyo de una quinta ontología.

La asignatura 'Arquitectura de Sistemas', impartida por el departamento de Ingeniería Telemática de la Universidad Carlos III de Madrid, tiene como uno de sus objetivos el aprendizaje del lenguaje de programación C para diseñar sistemas software. Los estudiantes de la asignatura a menudo cometen errores de compilación. El compilador ofrece diferentes mensajes ante dichos errores, pero en multitud de ocasiones es difícil para ellos detectar las causas de dichos errores. Para encontrar la solución al error, suele ser necesario que los alumnos revisen el material del curso o buscar información específica que pueda ayudarlos, pero esta búsqueda manual puede llevar mucho tiempo o no obtener los resultados esperados.

A raíz de todo ello, surgió la idea de realizar una herramienta software capaz de generar feedback eficaz que sirva de ayuda a los alumnos en la búsqueda de la solución a sus propios errores, y hacer así más efectivo el proceso de aprendizaje. La ayuda aparecerá en forma de conocimiento sobre el lenguaje de programación C, concretamente se desea ofrecer recomendaciones individualizadas e información personalizada en función de los errores de compilación que el alumno ha generado, que le oriente en su corrección.

Para la creación del software descrito, se utilizarán técnicas de Web Semántica. Algunas motivaciones para utilizar estas tecnologías son la interoperabilidad con otros recursos de la Web, la posibilidad de utilizar herramientas ya existentes en estas tecnologías o la capacidad de modelado que posibilitan.

1.2. Objetivos

En este proyecto se pretende realizar un tutor inteligente que recomiende recursos educativos (contenidos y ejercicios) para ayudar a los alumnos a conocer las causas de diferentes

errores de compilación que cometen cuando programan en C. Para lograrlo, se han identificado varios objetivos que serán explicados de forma detallada a continuación.

Como primer objetivo se encuentra el de modelar de manera formal una serie de dominios relacionados con la programación en C. Para esto, se deben elaborar un conjunto de ontologías utilizando técnicas de Web Semántica. En concreto, en este proyecto se han definido las siguientes ontologías:

- Una ontología que recoge los errores más comunes que los alumnos cometen mientras programan.
- Una ontología por cada uno de los errores anteriores, que refleje todas las posibles causas que provoca el error.
- Una ontología sobre el propio dominio de programación en C. Esta ontología también servirá para reflejar las conexiones entre las causas del error y la propia área de conocimiento, ya que incluye todos los conceptos relacionados con los errores posibles.

En segundo lugar, se debe realizar una anotación de los recursos educativos según los conceptos definidos en las ontologías, usando para ello RDF. Mediante la definición de un entorno Jena se hace posible la comunicación con la base de datos SDB donde se almacenan las ontologías y la anotación de los recursos.

Un tercer objetivo consiste en el desarrollo de una aplicación Java que incluya un razonador capaz de, a partir de los errores de compilación obtenidos y un diálogo con el alumno, ir recorriendo las ontologías e identificar la causa de error. Una vez se tiene identificada la causa de error, la aplicación debe ofrecer conocimiento que ayude al alumno a comprender y corregir sus propios errores, facilitando así el aprendizaje. Concretamente, el feedback que ha de generar la aplicación se debe componer de:

- Conocimiento sobre ciertos conceptos relacionados con la causa y definidos en la ontología sobre el propio dominio de programación en C.
- Un listado de recursos web educativos que ayuden al alumno en la corrección y comprensión de la causa de sus propios errores.

Además, la aplicación debe ser eficaz, y devolver información y conocimiento realmente útil para el alumno. La interfaz debe permitir una interacción sencilla y agradable.

Finalmente, como la asignatura para la que se pretende diseñar la aplicación consta de una Máquina Virtual con ciertas herramientas necesarias para generar un entorno de trabajo adecuado, se deberá integrar en ella para que sea fácilmente accesible a los alumnos. Además se debe tener en cuenta que esta aplicación no deja de ser una herramienta de ayuda, por lo que el alumno debe poder elegir si desea usarla o no.

1.3. Planificación

A continuación se enumeran de manera secuencial todas las tareas que se han planificado para la realización del proyecto:

1. Documentación y estudio del concepto de Web Semántica.
2. Elección de tecnologías.
3. Estudio sobre el lenguaje de programación C e identificación de los conceptos más importantes.
4. Estudio de todas las posibles causas que puedan provocar cada uno de los errores de compilación que se van a tratar.
5. Creación de todas las ontologías con el programa 'Protege'.
6. Etiquetado de todas las páginas web que conforman los apuntes de la asignatura 'Arquitectura de Sistemas'.
7. Generar un entorno de trabajo adecuado mediante la instalación de todos los programas, herramientas y bibliotecas necesarios.
8. Diseño e implementación de la aplicación en Java utilizando el entorno jena y la base de datos sdb para consulta y visualización de los datos.
9. Creación de casos de prueba para poder evaluar en trabajos futuros si el feedback con el conocimiento y los enlaces a páginas web devueltos por la aplicación, son útiles y reflejan la causa específica de los error de compilación del alumno.
10. Integración de la aplicación en la Máquina Virtual que se usa en la asignatura.
11. Preparación de toda la documentación del proyecto.

2. ESTADO DEL ARTE Y TECNOLOGÍAS UTILIZADAS

2.1. Estado del arte

A lo largo de este capítulo y con el objetivo de poner en contexto el trabajo realizado, se presentará la Web Semántica y se explicará el concepto de ontología. Además se estudiará su aplicación en el campo de la educación y se abordará el problema de la transición de la web actual a una Web Semántica.

También se hará una breve presentación de los sistemas de recomendación de información basado en realimentación, pues es un caso particular del uso que se le puede dar a la Web Semántica en educación, y el que se desarrolla en este proyecto.

2.1.1. La Web Semántica

En la actualidad, la principal herramienta de búsqueda de información es Internet. En los recursos web, la mayor parte de la información está representada en lenguaje natural para que sea fácilmente comprensible por los usuarios. Aunque esta simplicidad ha favorecido al éxito y crecimiento de la web actual, puede generar problemas a la hora de recuperar información y de que esa información pueda ser procesada de manera inteligente por las máquinas.

Como el contenido de la información no está estructurado en general; es decir, no está descrito o caracterizado de ninguna forma que sea directamente interpretable por los sistemas de información, nuestras máquinas no son capaces de identificar el contenido ya que no pueden entender e interpretar el significado del lenguaje natural.

Debido a ello, por ejemplo los motores de búsqueda son muy sensibles al vocabulario empleado y sus resultados pueden ser poco precisos y no satisfacer las necesidades de los usuarios. Esto ocurre cuando en la búsqueda empleamos palabras que poseen varios significados, o sinónimos en lugar de la palabra que aparece en el contenido del recurso. Además, se añade el problema de que el usuario no tiene información sobre la veracidad de los sitios web recuperados por el buscador.

Una solución a este problema es la llamada Web Semántica, propuesta por Tim Berners-Lee [2] e impulsada por el W3C [1], cuyo logo se puede ver en la figura 2.1. Se trata de una web

[1] W3C (World Wide Web Consortium) [3]: consorcio internacional que produce recomendaciones para la World Wide Web. Fue creado el 1 de octubre de 1994 por Tim Berners-Lee, creador de la WWW

extendida, dotada de mayor significado y formada por una información mejor definida, en la que los usuarios de Internet podrán recuperar información de forma más rápida y sencilla, y en la que las máquinas pueden razonar sobre esos datos. Esto se conseguiría dotando a los recursos web de un significado semántico mediante la incursión de 'metadatos'.

Figura 2.1: Logo W3C Semantic Web. Fuente [3]

Estos metadatos habilitarán una web inteligente donde los recursos no sólo tendrán texto plano, sino que estarán anotados con información semántica que: describa y clasifique la información que contienen, defina relaciones entre recursos, cree relaciones de conceptos, etc.

De este modo las máquinas ya sí entenderán el contenido de la información y podrán manipularla y efectuar un procesamiento automático mucho más profundo. Pero no sólo del contenido textual, sino de cualquier información que circule por Internet que utilice estas tecnologías, ya sean imágenes, audios, servicios web, o incluso sólo metadatos.

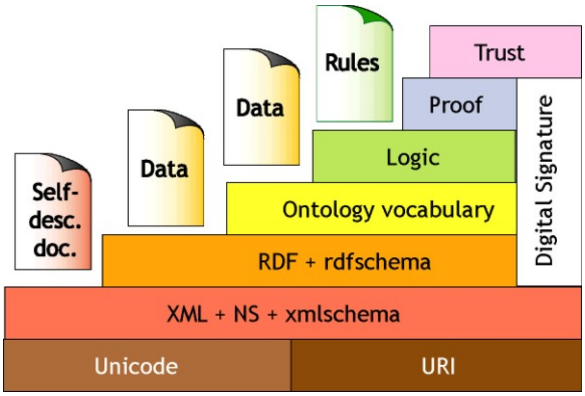


Figura 2.2: Arquitectura de la Web Semántica. Fuente [3]

La arquitectura de la Web Semántica se puede observar en la figura 2. Se divide en varias capas, cada una de las cuales está basada y apoyada en las inferiores aportándolas nuevas y distintas funcionalidades. A continuación se explican ca da una de estas capas de forma detallada:

- Unicode: estándar de codificación de caracteres cuya finalidad es facilitar el tratamiento, transmisión y visualización de textos en múltiples idiomas y lenguajes sin que aparezcan símbolos no válidos. Representa el alfabeto.
- URI (Universal Resource Identifier): sistema de direccionamiento que identifica inequívocamente a cada recurso web, permitiendo así su localización.
- XML + NS + xmlschema: tecnologías que hacen posible que los agentes puedan entenderse entre ellos. Para lograrlo crean documentos uniformes en un formato común, no propietario e independiente de la fuente.

Esta capa junto con las dos anteriores definen los elementos web básicos, los cuales servirán como base a los modelos RDF que conforman la Web Semántica.

- RDF (Resource Description Framework) + rdfschema: lenguaje capaz de dotar a cada recurso de la red de una lógica y un significado que permitan a los ordenadores conocer el significado de la información contenida en ellos.

  La mayoría de los sitios web actuales están editados o en lenguaje HTML o en XML, los cuales definen etiquetas que permanecen ocultas en la visualización normal de los navegadores y que contienen información sobre el contenido de la página, enlaces hacia otras páginas, formatos de letra, color, párrafos, imágenes, vídeos, etc. Pero para desarrollar una Web Semántica hacen falta otros lenguajes más completos y que permiten una descripción más detallada del documento y su contenido. Para ello el W3C aconseja RDF, cuya sintaxis está basada en XML.

  Las limitaciones de RDF surgen por no ofrecer ninguna especificación del vocabulario que se debe usar para describir metadatos. Para solucionarlo el W3C propuso RDF Schema (RDFS) como extensión semántica, que es un lenguaje que completa el modelo RDF básico proporcionando información sobre la interpretación de las sentencias para así alcanzar interoperabilidad semántica en el contexto de la Web Semántica.

  Se puede entender como un lenguaje muy básico para definir ontologías, proporcionando los elementos básicos para la descripción de vocabularios. RDFS no sólo provee un sistema para definir tipos, sino que conjuntamente se pueden usar una serie de tipos básicos ya definidos permitiendo que usuarios de distintos dominios compartan vocabularios.

  El problema es que carece de poder expresivo para representar restricciones lógicas, lo que se necesitaría para el desarrollo de la Web Semántica. Por esta razón, existe una capa lógica por encima.

  En resumen, la capa referente a RDF incluye metadatos a los recursos web, aportándoles una descripción sobre el contenido de los datos y cierta información semántica sobre los mismos.

- Ontologías: una ontología es una definición formal del dominio del conocimiento que deseamos representar semánticamente. Nos permiten describir conceptos básicos del dominio, clasificarlos y marcar las diferentes relaciones que los unen con otros conceptos.

  Para crear ontologías, se necesitan lenguajes semánticos formales (basados en RDF) con suficiente capacidad expresiva y de razonamiento como para representar todo el dominio del conocimiento. Además, estos lenguajes deben ser estandarizados y formalizados para que su uso sea universal, reutilizable y compartible.

- Lógica: como las ontologías se expresan en un lenguaje basado en la lógica, se pueden definir razonamientos que, aplicados a las reglas expresadas en la ontología, den como resultado los datos deseados.

- Pruebas: intercambio de pruebas escritas en el lenguaje unificador.

- Confianza: se debe ser cuidadoso con los datos recuperados de la web hasta que no se verifique que provienen de una fuentes de información fiable. Una de las ventajas que aporta la Web Semántica, es que esta comprobación es posible de forma automática. Para ello se usan técnicas de encriptado de datos.
- Firma digital: la Web Semántica utiliza XML Signature WG como firma digital.

En resumen, la Web Semántica se caracterizará por ser capaz de: comprender el contenido de los recursos web, procesarlos, obtener las fuentes más fiables e incluso relacionar la información contenida en páginas hoy aisladas. Todo ello permite crear software capaz de realizar deducciones lógicas y tomar decisiones con un cierto grado de autonomía, para entender de forma exacta por ejemplo las demandas de información de los usuarios.

Hay expertos, como Abian [4] y el propio Berners-Lee [2], que defienden que la Web Semántica es el futuro de la web y puede ser una pieza importante para el progreso de la sociedad de la información.

A pesar de que esta tecnología está aún en proceso de desarrollo, ha despertado gran interés tanto en empresas privadas (IBM, Microsoft, Sun, Oracle tre otras) como en el sector público (programas marco EU-IST en Europa y DARPA en EE.UU.), e incluso en el mundo académico [5]. Actualmente, se está estudiando en importantes centros de investigación y universidades de todo el mundo y en pocos años se han obtenido avances importantes que se publican en publicaciones especializadas en este área como 'Journal of Web Semantics' [6] o la sección 'The Semantic Web' del 'Electronic Transactions on Artificial Intelligence' [7]. Como prueba de que ya existe una comunidad investigadora considerable se debe mencionar al congreso internacional 'International Semantic Web Conference' [8].

Transición de la web actual a la Web Semántica

La transición de la web actual a la Web Semántica no se hará de manera inmediata, pues habría que introducir metadatos a las millones de páginas web que ya forman parte de la red [4].

Por lo tanto se nos plantea el problema de cómo realizar una transición óptima teniendo en cuenta el inmenso volumen de recursos que habría que modificar. Las estrategias más viable, combinan una pequeña parte de trabajo manual con la automatización del resto del proceso [4]. Las acciones principales que se pueden realizar de forma automática especificadas en [5] son: el mapeo de la estructura de bases de datos a ontologías, la adaptación de los metadatos y estándares de clasificación ya presentes en la web y la extracción automática de metadatos a partir de texto y recursos multimedia.

Una dificultad importante que surge al implantar la Web Semántica de forma práctica, es la de conseguir converger a una representación común a pesar de que cada parte del sistema conlleva sus propias peculiaridades, por lo que es importante que la Web Semántica guarde una compatibilidad con la tecnología actual [5].

Otro problema surge por la falta de totalidad de ontologías, por lo que desarrollar todas las ontologías que faltan y son necesarias supondría un gran trabajo y esfuerzo [9]. Además,

si tenemos en cuenta que a algunos conceptos no siempre se les puede imponer una misma definición porque dependen del contexto o del punto de vista que se les de, la creación de ontologías se complica. Una solución posible a esta complicación es la de compartir ontologías para las áreas comunes, como se pretende en [10].

A pesar de estas complicaciones, se ha avanzado mucho con las herramientas, los estándares y la infraestructura que serían necesarios para una transición hacia la Web Semántica, e incluso ya se han desarrollado proyectos y experiencias piloto para ponerlas a prueba [Haustein 2002].

La web semántica en educación

Desde 1990, la educación basada en la Web se ha convertido en una rama muy importante de la tecnología educativa. Los entornos e-Learning ofrecen herramientas completamente virtualizadas que, mediante una interacción con el alumno, proporcionan recursos didácticos para que los estudiantes aprendan de forma autónoma.

El e-Learning abarca un amplio conjunto de aplicaciones y procesos, como aprendizaje basado en la web, aulas virtuales y colaboración digital. Incluye la entrega de contenido a través de Internet, LAN/WAN, audio y video, transmisión vía satélite, TV interactiva, y CD-ROM (Kaplan-Leiserson, 2000).

Las ventajas que supone el uso de e-Learning se recogen en [11]. Por un lado los alumnos tienen un acceso personalizado a los recursos educativos, logrando un progreso más rápido que con el aprendizaje tradicional. El material que se les ofrece es coherente y pueden tener acceso a él en cualquier momento y en cualquier lugar. Por otro lado, los profesores también se benefician de diferentes posibilidades como: barata y eficiente distribución de los materiales del curso, fácil y rápido mantenimiento de los recursos educativos y posibilidad de añadir diversos elementos didácticos y herramientas que refuercen el aprendizaje como audio, video, enlaces directos a otros recursos web que complementan la información, simulaciones, etc.

Como se defiende en [12], la reusabilidad e interoperatibilidad en educación es esencial. El e-Learning debe garantizar el intercambio entre plataformas y crear recursos de calidad que puedan ser reutilizados en diferentes contextos. Para que esto sea posible, se han desarrollado diferentes estándares y especificaciones de e-learning. Como se describe en el 'Libro de Buenas Prácticas de e-learning' [13], las instituciones y organizaciones más involucradas en el proceso de estandarización son el IEEE LTSC (Learning Technology Standards Committee), ISO (International Standards Organization), IMS, AICC (Aviation Industry CBT Committee) y ADL (Advanced Distributed Learning). Además, se han elaborado multitud de trabajos cuyos resultados han sido utilizados para elaborar especificaciones y estándares como Prometeus (PROmoting Multimedia access to Education and Training in the EUropean Society), GESTALT (Getting Educational Systems Talking Across Leading edge Technologies) y Ariadne I y II.

Una de las funcionalidades más importantes de los sistemas de e-Learning [14], es la de ofrecer una enseñanza individualizada en tiempo real, haciendo que el aprendizaje sea más efectivo. Se debe ofrecer al alumno un material personalizado dependiendo de sus propias

características: nivel de conocimientos, estilo de aprendizaje, idiomas, roles asumidos, tipo de contenidos, acciones previas realizadas, progreso en el proceso de aprendizaje, dispositivo con el que acceder a la información, etc.

La Web Semántica puede ser utilizada en la gestión o acceso a los contenidos de manera guiada por el conocimiento. De esta forma se conseguirá filtrar el contenido ofreciendo información adaptada a diferentes entornos de aprendizaje o usuarios. Dentro del campo educativo, ya existen trabajos que demuestran que el uso de la Web Semántica como sistema adaptativo es posible.

El sistema TANGRAM [15], basándose en una serie de ontologías, permite la descomposición de los recursos en pequeñas unidades de contenido que serán posteriormente montadas dinámicamente según necesidades del usuario, construyendo así un contenido personalizado.

En [16] se propone el uso de mecanismos de razonamientos y anotaciones RDF para llevar a cabo la adaptación. Define varias ontologías para modelar: los recursos, las relaciones entre los recursos, los usuarios y la interacción posible entre el usuario y la representación hipertextual.

Como indica Jason Ohler en [17], la Web Semántica se puede usar para realizar búsquedas más efectivas de los recursos educativos. En lugar de una simple lista de enlaces, se puede generar un informe multimedia con diversas fuentes de información como recursos web, artículos en repositorios científicos, capítulos determinados de libros electrónicos, diálogos en blog, videos, etc. El objetivo es que el alumno no tenga que realizar una búsqueda manual y gaste menos tiempo en obtener los recursos deseados. Un ejemplo de los proyectos desarrollados hasta ahora que usan técnicas de Web Semántica en búsquedas de recursos, es el 'Watchdog Courseware'[18] formado por un conjunto de herramientas basadas en ontologías.

Una última aplicación de la Web Semántica en educación, es el posible uso de la propia ontología como recurso didáctico. Se pueden presentar al alumno como un esquema que define el dominio del conocimiento de interés a través de un conjunto de conceptos relacionados entre sí y, como suelen ser el resultado de un amplio consenso de expertos en la materia, proporcionan información fiable y objetiva [11]. Al mismo tiempo, los expertos pueden utilizar ontologías para plasmar su propia comprensión del dominio.

Las herramientas actuales para el desarrollo de ontologías, requieren de cierto grado de especialización que no hace viable la creación de ontologías por parte de personas que, aunque expertas en el dominio, no conozca la ingeniería ontológica.

Existen distintas bibliotecas donde se recogen ontologías útiles desarrolladas hasta el momento como DAML Ontology Library [19] y Protege Ontology Library [20].

Además, existen proyectos para la elaboración de ontologías sobre un área de conocimiento concreto. El proyecto WOP [21] tiene como objetivo la creación de una ontología (llamada ODOL) sobre diseño de software orientado a objetos.

El principal problema que hay que solventar, es la falta de ontologías de dominio para desarrollar y organizar el contenido didáctico. Para poder desarrollar ontologías educativas

válidas se necesita un gran esfuerzo y trabajo de colaboración, para lo que ya se han comenzado diversas iniciativas.

El portal O4E [10] ha desarrollado una ontología O4E para indexar trabajos y ontologías sobre aplicaciones de la Web Semántica en educación. Otras funciones del portal es permitir el acceso a la comunidad O4E, workshops y conferences y ofrecer una lista de herramientas para la elaboración de ontologías y anotación de recursos.

2.1.2. Sistemas de recomendación en educación

Un sistema de búsqueda de información se compone de herramientas informáticas que permiten acceder a una información determinada mediante estrategias de búsqueda específicas. Dicha información ha debido ser estructurada previamente para que, mediante consultas en un lenguaje de interrogación adecuado, se pueda acceder a la información requerida.

Los problemas que nos pueden surgir al analizar la información recuperada por estos sistemas son:

- Existen documentos que podían ser útiles y no se han recuperado porque la estrategia de búsqueda ha sido demasiado específica.
- Hay documentos recuperados por el sistema que no son relevantes y no deberían haber sido devueltos.
- Se han recuperado demasiados documentos, lo que puede provocar que el usuario se sature de información y no vea efectivo el sistema de recuperación de información. Para solventar este problema se aconseja crear un ranking; es decir, colocar los documentos por orden de importancia para que el más relevante se muestre el primero y el menos importante el último.

Teniendo en cuenta estos problemas, no es de extrañar que la calidad de los sistemas de recuperación de información se mida teniendo en cuenta los siguientes criterios básicos:

- Consistencia: capacidad de establecer ecuaciones de búsqueda sobre términos admitidos, controlando los errores.
- Exhaustividad: capacidad de recuperar los documentos con información relevantes de entre la totalidad de documentos almacenados en la base de datos.
- Tasa de relevancia: coeficiente resultante de dividir el número de documentos relevantes recuperados, en los cuales la información contenida es útil al usuario, entre el número total de documentos almacenados en la base de datos.
- Tasa de acierto: coeficiente resultante de dividir el número de documentos relevantes recuperados entre el número total de documentos relevantes almacenados en la base de datos.
- Tasa de pertinencia: coeficiente resultante de dividir el número de documentos pertinentes recuperados entre el número total de documentos almacenados en la base de datos.

- Tasa de precisión: coeficiente resultante de dividir el número de documentos relevantes recuperados, sobre el número total de documentos almacenados en la base de datos.

Los recomendadores de información deben predecir y anticiparse a las necesidades de los usuarios para poder generar una respuesta eficiente [22]. Para ello se deben recoger datos sobre el usuario para elaborar su perfil y, mediante comparación con otros de referencia, ser capaz de recuperar información personalizada y de interés para ese usuario específico. Estos sistemas se pueden entender como extensiones de los sistemas de búsqueda de información.

Lo más importante para poder realizar una buena recomendación de información es desarrollar un algoritmo adecuado. Ejemplos de tecnologías usadas en algunos algoritmos de recomendación existentes son: demographic, knowledge-based, Community-based, and hybridizations [23].

Además, los sistemas de recomendación deben tener conocimiento sobre las fuentes de información, delimitar las búsquedas según ciertos criterios, desarrollar habilidades para evaluar los resultados y determinar un acceso a la información que respete los principios éticos y legales.

Respecto al uso de recomendadores en educación, ya se han llevado a cabo distintas iniciativas. EATEL (European Association of Technology Enhanced Learning) [24] es una asociación que apoya la investigación de distintos laboratorios, patrocina eventos sobre TEL[1] y permite el intercambio de conocimientos. Los sistemas de recomendación son una de las líneas de investigación de la asociación, ya que estos sistemas ofrecen una prometedora forma de facilitar el aprendizaje y la enseñanza.

En [25] se realiza una comparativa de algunos de los sistemas de recomendación en educación existentes que incorporan distintas formas de incluir información contextual para llevar a cabo su proceso de recomendación. Entre ellos se encuentran: Zhao et al, COLDEX, Lehsten et al, Rogers et al, Perkam, TANGO, CALS, PALLAS, APOSDLE, MobiLearn, etc.

Sistemas de recomendación basados en realimentación

Un sistema de realimentación (en inglés feedback) es un mecanismos de control de sistemas dinámicos donde parte de los datos de salida se redirige a la entrada, regulando así su comportamiento. En el caso concreto de recomendadores basadas en realimentación, los datos que vuelven a la entrada son proporcionados por el usuario y van refinando la búsqueda hasta encontrar el resultado adecuado. La realimentación se puede generar de dos formas:

- De forma explicita: solicitando información al usuario de forma directa. La interacción con el usuario permite la formulación de consultas eficientes que permiten el intercambio de información mediante software entre el sistema y el usuario, haciendo más productivas las técnicas de búsqueda.

- De forma implícita: recogiendo datos de forma transparente para el usuario, a través de las acciones que el usuario realiza con su entorno.

[1]TEL (Technology Enhanced Learning): conjunto de tecnologías que tienen como objetivo diseñar, desarrollar y probar nuevos métodos y herramientas que mejoren las prácticas de aprendizaje de los alumnos.

En la mayoría de recomendadores, se necesita hacer uso de una interfaz gráfica para iniciar la recuperación o para visualizar los resultados [25]. Pero si además se basan en realimentación y necesitan una interacción con el usuario, la interfaz adquiere un papel mucho más importante.

Hay que tener en cuenta que el ser humano tiene una capacidad limitada de procesar información. Es aconsejable ir mostrando en todo momento las elecciones ya tomadas para que el usuario no las tenga que recordar y ofrecer mensajes informativos orientados al usuario para que sepa como interactuar con el sistema.

Otras mediadas a tomar en cuenta para crear una interacción agradable para el usuario son: garantizar unas respuestas de tiempo adecuadas, facilitar la reversibilidad de las acciones, usar adecuados recursos de fuente de letra, color, audio, etc, simplicidad del interfaz y prever errores que puede cometer el usuario para buscar puntos exactos donde darle el control. Además, un hecho que no se puede pasar por alto es que todos los usuarios tendrán habilidades comunes, pero habrá otras que variarán según la persona.

Como ejemplo de recomendador basados en realimentación se encuentra el sistema LOCO (Learning Object Context Ontologies) [26]. Se trata de una sistema basado en Web Semántica que, mediante la interacción con el usuario, proporciona realimentación educacional capaz de generar recursos adaptados a su perfil. Para ello, define diversas ontologías: una ontología del contexto de aprendizaje , una ontología para modelar a todos los participantes en el proceso de aprendizaje, una ontología que identifica cada uno de los recurso, una ontología para representar formalmente instrumentos de evaluación y una ontología sobre el propio dominio de conocimiento de los contenidos académicos.

2.2. Tecnologías utilizadas

A continuación se presentan las tecnologías que han sido utilizadas en este proyecto.

2.2.1. RDF

RDF es el acrónimo de 'Resource Descripción Framework' [22]. Fue creado en agosto de 1998 y recomendado por W3C en 1999 como estándar para la Web Semántica. Su objetivo principal es definir un protocolo para la descripción de recursos web que garantice la compatibilidad entre los diversos sistemas de metadatos que proporcionan información descriptiva simple sobre el contenido del recurso. Se decidió utilizar el lenguaje XML [27] como sistema de comunicación, ya que es uno de los más implantados en la web actual.

Los principios básicos de RDF son la interoperatividad de metadatos, garantizar la multiplataforma para que sea independiente de software y/o sistema operativo y flexibilidad para que sea capaz de describir cualquier tipo de información. Actualmente RDF se ha convertido en un estándar popular y de amplio uso en la comunidad Web Semántica.

RDF usa tripletas para la descripción simple de recursos web. Las tripletas son expresiones compuestas de:

- Sujeto: se refiere al recurso web sobre el que se realiza la descripción.
- Predicado: propiedad o relación que se desea establecer sobre el recurso.
- Objeto: valor de la propiedad o el otro recurso con el que se establece la relación.

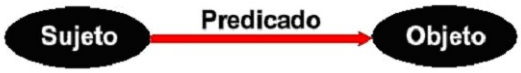


Figura 2.3: Grafo que representa los elementos de una tripleta

La figura 2.3 recoge una descripción gráfica de las partes de una tripleta y sus relaciones. En la figura 2.4 se puede consultar un ejemplo gráfico de tripleta que define que un recurso web concreto (www.it.uc3m.es/pointer_es.html) contiene información sobre punteros.

Figura 2.4: Ejemplo de una tripleta RDF de forma gráfica

RDF está basado en la idea de identificar recursos usando URIs que puede ser creado por nosotros o que esté ya definido en alguna otra parte. Tanto el sujeto como el predicado se indican mediante una URI, mientras que el objeto puede ser una URI o un literal.

Existen modelos de metadatos que definen espacios de nombres mediante URIs que identifican diferentes elementos del lenguaje RDF y que se pueden usar a la hora de especificar nuestras propias triplas. Un ejemplo de ello es el Dublín Core [28] elaborado por la DCMI [1].

```
<http://www.it.uc3m.es/pointers_es.html><http://purl.org/dc/elem
ents/1.1/subject>"Punteros"
```

Figura 2.5: Ejemplo de una tripleta en RDF

El mismo ejemplo de tripleta de la figura 2.4 pero usando lenguaje RDF sería el que se muestra en la figura 2.5. Como se puede observar, para indicar la propiedad de 'tiene información sobre' se hace uso de la URI definida en el Dublín Core que representa a la propiedad 'subject'. Esta se puede emplear para expresar las claves o frases que describen el título o el contenido del recurso.

Se han definido diferentes formas sintácticas para la formulación escrita de RDF, pero la más extendida es la basada en XML y es por ello que RDF se presenta a menudo como una extensión de este lenguaje. Para describir tripletas se usa una sintaxis serializada asociada en XML y basada en los estándares de URI y Unicode.

```
1.    <rdf:RDF
2.      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.      xmlns:dc="http://purl.org/dc/elements/1.1/">

4.          <rdf:Description rdf:about="http://www.it.uc3m.es/
pointers_es.html">
5.                  <dc:subject>Punteros</dc:subject >
6.          </rdf:Description>

7.    </rdf:RDF>
```

Figura 2.6: Ejemplo de una tripleta en XML

En la figura 2.6 se muestra como quedaría la tripleta de ejemplo de las figuras anteriores, una vez escrita en lenguaje XML.

En las líneas 2 y 3 se definen prefijos de los identificadores de cada espacio de nombres. Con ello logramos que la expresión 'dc:subject' donde se ha definido con anterioridad que dc='http://purl.org/dc/elements/1.1/', sea igual a 'http://purl.org/dc/elements/1.1/subject'

La línea 4 indica que estamos describiendo un recurso, concretamente el indicado por el atributo 'rdf:about'. La propiedad y su valor se reflejan en la línea 5.

Las líneas 6 y 7 cierran los anteriores elementos siguiendo una estructura de etiquetas.

2.2.2. SPARQL

SPARQL es el acrónimo de 'Protocol and RDF Query Language' [29]. Es un lenguaje de consulta para la recuperación de información a través de RDF. Desde 2005 está en proceso de estandarización por W3C y en 2007 lo propusieron como estándar para desarrollar Web Semántica. Actualmente RDF está siendo usado para proporcionar un medio de integración entre fuentes de información heterogéneas.

Su desarrollo surgió como necesidad ya que si suponemos que cada recurso web tiene un conjunto de metadatos que le definen, nos encontramos con un montón de ficheros RDF que hay que almacenar y manejar dentro de una gran base de datos. Por lo que se hace necesaria una herramienta que nos permita realizar consultas a dicha base de datos centrándose únicamente en los metadatos, para optimizar así las búsquedas de este tipo de información.

El lenguaje SPARQL ha sido diseñado para su uso a escala de la Web ya que es independientemente del formato. Además, la creación de una sola consulta independientemente de la tecnología de la base de datos o el formato utilizado para almacenar los datos, ofrece mejores resultados y menor coste.

Este lenguaje presenta 3 especificaciones orientadas, cada una de ella, a una funcionalidad:

[1]DCMI (Dublin Core Metadata Initiative): organización cuya finalidad es desarrollar estándares interoperables de metadatos y promover el desarrollo de vocabularios especializados de metadatos para describir recursos.

- SPARQL Query Language: componente principal de SPARQL. Explica la sintaxis del lenguaje de consulta para la composición de sentencias y su concordancia.
- SPARQL Protocol for RDF: formato utilizado para las respuestas o devolución de los resultados de las búsquedas, a partir de un esquema RDF-XML.
- SPARQL Query Results XML Format: describe el medio para el transporte de consultas y respuestas entre los clientes y los procesadores. Utiliza WSDL para definir protocolos remotos para la consulta de bases de datos basadas en RDF.

El lenguaje de consulta SPARQL está basado en comparación de triples, que son como tripletas RDF pero que además incluyen la opción de manejar sólo una variable que se encuentre en las posiciones del sujeto, predicado u objeto, y no obligatoriamente la tripleta entera. Las consultas SPARQL devolverán las tripletas o conjunto de variables de entre todos los que forman el grafo, que coincidan con el patrón de búsqueda.

Las consultas se forman mediante tres componentes principales: URIs, literales descritos como una cadena de caracteres entre comillas dobles y variables globales procedentes del lenguaje RDF que deberán llevar los prefijos '?' o '$' y estar separadas por un caracter de punto y coma. El comienzo de toda consulta SPARQL se marca por la palabra clave SELECT que sirve para marcar los datos que deben ser devueltos.

Las consultas pueden incluir otras restricciones no obligatorias que devuelvan una respuesta mucho más concreta. Por ejemplo se pueden incluir: prefijos (mediante la palabra clave PREFIX), especificación de los datos sobre los que realizar la consulta (FROM), especificación de tripletas concretas sobre las que realizar la consulta (WHERE), ordenación de resultados (ORDER), límite de resultados (LIMIT)... O también se pueden usar palabras claves que provocan que la consulta devuelva diferentes elementos como: un grafo RDF que describe los recursos encontrados (DESCRIBE), un boolean indicando si los patrones de la consulta coinciden o no (ASK), un grafo RDF construido por la sustitución de variables (CONSTRUCT)...

Un ejemplo sobre la sintaxis básica de una consulta SPARQL sería el que se muestra en la figura 2.7, donde se solicitan todos los sujetos (representados mediante la variable recurso) que tienen el valor 'Puntero' para la propiedad o predicado 'subject'. O lo que es lo mismo, se solicitan todos los recursos web cuyo contenido trata sobre punteros.

```
SELECT ?recurso
WHERE {
?recurso <http://purl.org/dc/elements/1.1/subject>"Puntero"
}
```

Figura 2.7: Ejemplo de consulta SPARQL básica

Por tanto, si suponemos que tenemos como datos sobre los que realizar la consulta de la figura 2.7, los datos RDF que se muestran en la figura 2.8, el resultado a la consulta sería el mostrado en la figura 2.9.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">

     <rdf:Description
rdf:about="http://www.it.uc3m.es/labas/course_notes/pointers_es.
html">
          <dc:subject>Puntero</dc:subject>
     </rdf:Description>

     <rdf:Description rdf:about="http://www.it.uc3m.es/
data_types_es.html">
          <dc:subject>Tipo_dato</dc:subject>
     </rdf:Description>

     <rdf:Description rdf:about="http://www.it.uc3m.es/
data_types_es.html">
          <dc:subject>Tabla</dc:subject>
     </rdf:Description>

</rdf:RDF>
```

Figura 2.8: Ejemplo de etiquetado de datos RDF

```
recurso
http://www.it.uc3m.es/pointers_es.html
```

Figura 2.9: Ejemplo de respuesta a una consulta SPARQL básica

2.2.3. OWL

OWL (Ontology Web Language) [30] es un lenguaje de marcado cuyo objetivo principal es definir ontologías muy completas que definan un dominio de conocimiento y conseguir un contenido web más accesible a las máquinas. Ha sido desarrollado recientemente para enriquecer aún más a RDF/RDFS y superar su insuficiente poder expresivo, por lo que está construido sobre RDF y codificado en XML.

En comparación con RDFS, OWL permite expresar relaciones mucho más ricas y, por tanto, disponer de capacidades de inferencia mejoradas.

Algunas de las posibilidades adicionales que ofrece OWL frente a sus predecesores son:

- Definir y describir clases mediante restricciones sobre propiedades, valores o cardinalidad.
- Permitir herencia de clases.
- Definir y describir objetos que pertenecen a una clase.
- Definir y describir propiedades que pueden ser inversas, transitivas, simétricas funcionales o funcionales inversas.

- Permitir intersección, unión y complemento.
- Crear cuantificadores existenciales y universales.
- Crear instancias de elementos, tipos de datos y anotaciones.
- Clases enumeradas.

El principal problema que presenta OWL es que la escalabilidad a una web extensa se hace muy compleja. Esto se debe a que para realizar razonamientos sobre ontologías soportadas por OWL, se necesita lógica de descripciones y aún no se ha definido ningún algoritmos que garantice la complejidad computacional necesario. Es posible que no se encuentren todas las conclusiones válidas o que las que se encuentre no sean totalmente correctas o que se encuentren todas en tiempo de razonamiento válido.

Para solventar este problema, se crearon tres variantes de OWL dependiendo de la relación expresividad-computabilidad que se desee soportar:

- OWL Lite: es la variante más simple. Sus ventajas principales son que es fácil de entender y fácil de implementar, pero a cambio debemos pagar un precio de pérdida de expresividad.
- OWL DL: es la versión aconsejada para aquellos usuarios que necesiten el máximo de expresividad sin perder la capacidad de ser computable. El problema es que sólo se puede utilizar bajo ciertas restricciones.
- OWL Full: permite la máxima expresividad y ofrece la libertad sintáctica de RDF pero carece de garantías en cuanto a la propiedad de ser computable.

Este lenguaje también se define en dos versiones: OWL 1 creado en 2004 y OWL 2 que surgió en 2009. La última versión añade nuevas funcionalidades como una nueva sintáxis, cadenas de propiedades, enriquecimiento de los tipos de datos y de los rangos de datos... Los cambios también permiten convertir un grafo RDF, de nuevo a OWL.

La parte lógica de una ontología OWL 2 contiene tres categorías sintácticas:

- Entidades: se refiere a las clases, propiedades, individuos y otros elementos esenciales que forman parte de conceptos del dominio que se está modelando.
- Expresiones: nociones complejas del dominio.
- Axiomas: declaraciones que se afirman que son verdad en el dominio que se está modelando.

  Estos elementos se construyen utilizando identificadores de recursos internacionales (IRI).

2.2.4. JENA

Jena es un es un framework Java [31] para crear aplicaciones de Web Semántica desarrollado por Hewlett-Packard Development Company y basado en las recomendaciones W3C. Incluye software desarrollado por Apache Software Foundation.

Proporciona un conjunto de herramientas y bibliotecas que permitir leer, recorrer y modificar información en grafos tanto RDF como OWL desde un programa Java. Las últimas versiones de Jena han incorporado motores de razonamiento para las expresiones lógicas de OWL y permiten la consulta de los modelos mediante SPARQL. Además incluye un parser sintáctico.

Jena también permite el almacenamiento de ontologías tanto en RDF textual como en formato de base de datos, lo que es importante para grafos muy grandes y pesados. Para el almacemaniento en gestores de bases de datos, necesita drivers JDBC.

Su arquitectura se ha desarrollado a partir de diversos subsistemas, algunos de los cuales pueden funcionar como aplicaciones o bibliotecas por sí mismas. Además, se le han ido añadiendo a su código nuevas funcionalidades en forma de clases, paquetes, bibliotecas, aplicaciones u ontologías.

Los paquetes que conforman el API de Jena [32], con una breve descripción de su funcionalidad, son los que se muestran a continuación:

- com.hp.hpl.jena.datatypes: proporciona las interfaces principales que describen los tipos de datos en Jena.
- com.hp.hpl.jena.datatypes.xsd: completan el paquete anterior proporcionando nuevas definiciones de tipos de datos que únicamente son necesarias si se desea desarrollar un esquema XML.
- com.hp.hpl.jena.ontology: proporciona un conjunto de clases para el acceso y manipulación de ontologías en RDF.
- com.hp.hpl.jena.rdf.arp: parser RDF / XML.
- com.hp.hpl.jena.rdf.listeners: define implementaciones útiles de ModelChangedListener capaces de identificar todas las tripletas agregadas o quitadas, todos los objetos añadidos o eliminados, aceptar y/o ignorar todos los cambios, etc.
- com.hp.hpl.jena.rdf.model: para crear y manipular gráficos RDF.
- com.hp.hpl.jena.reasoner: razonador proporcionado por la arquitectura de Jena.
- com.hp.hpl.jena.reasoner.rulesys: motores de reglas simples para los modelos de inferencia de Jena.
- com.hp.hpl.jena.reasoner.rulesys.builtins: proporciona operaciones primitivas para los motores definidos en el paquete anterior.

- com.hp.hpl.jena.reasoner.transitiveReasoner: razonador capaz de identificar relaciones del tipo subPropertyOf y subClassOf.
- com.hp.hpl.jena.shared: incluye la clase JenaException que proporciona todas las excepciones específicas para Jena y el PrefixMapping interfaz para el uso de QNames.
- com.hp.hpl.jena.vocabulary: contiene la predefinición de clases y objetos constantes para clases y propiedades definidas en un vocabularios conocido.
- com.hp.hpl.jena.xmloutput: permite la escritura en RDF / XML.

2.2.5. SDB

SDB [33] es un componente de Jena para el almacenamiento de RDF y consultas mediante SPARQL.

Necesita una descripción específica del 'store' donde se van a almacenar los datos RDF, que se puede definir en un archivo o mediante código usando herramientas SDB. Un ejemplo de este archivo de configuración se puede consultar en la figura 2.10. En el se recoge toda la información sobre el formato, la carga y el acceso a dicha base de datos.

```
@prefix sdb:     <http://jena.hpl.hp.com/2007/sdb#> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ja:      <http://jena.hpl.hp.com/2005/11/Assembler#> .

# MySQL - InnoDB


<#store> rdf:type sdb:Store ;
    sdb:layout     "layout2" ;
    sdb:connection <#conn> ;
    sdb:engine     "InnoDB" ;      # MySQL specific
.

<#conn> rdf:type sdb:SDBConnection ;
    sdb:sdbType       "MySQL" ;    # Needed for JDBC URL
    sdb:sdbHost       "localhost" ;
    sdb:sdbName       "nombreBaseDatosSDB" ;
    sdb:sdbUser       "nombreUsuario" ;
    sdb:sdbPassword   "contraseña" ;
    sdb:driver        "com.mysql.jdbc.Driver" ;
    .
```

Figura 2.10: Archivo de configuración del store de la base de datos SDB

Las principales clases y paquetes del API de SDB que se encargan del acceso y consulta de la base de datos son las siguientes:

- SDBConnection: lleva a cabo la conexión con la base de datos, así como la desconexión de la misma (StoreDesc).
- SDBFactory: describe las clases necesarias para la correcta comunicación con la base de datos.

- DatasetStore: accede a la descripción del store para crear el modelo de datos RDF. Maneja un motor de búsqueda capaz de manejar consultas realizadas sobre un conjunto de datos RDF.
- GraphSDB: implementa una interfaz SDB basándose en la interfaz de datos que proporciona Jena.

SDB también proporciona herramientas para manejar la base de datos y poder trabajar con tripletas desde línea de comandos. Algunas de estas herramientas son:

- sdbconfig: inicializa la base de datos y añade índices.
- sdbload: carga datos en la base de datos.
- sdbquery: realiza consultas SPARQL.
- sdbtest: ejecuta un archivo de prueba que comprueba la conexión y la carga de datos.

2.2.6. Protégé

Protege [34] fue desarrollada por el grupo SMI (Stanford Medical Informatics) de la Universidad de Stanford, y actualmente se ha convertido en el editor de ontologías más popular.

Protégé posee su propio lenguaje interno para definir ontologías de manera eficiente, pero también permite trabajar con una gran variedad de formatos, incluyendo OWL, RDF y XML-Schema. Es capaz de procesar grandes ontologías de forma eficiente y utiliza el almacenamiento en caché para liberar memoria cuando es necesario.

Permite de forma cómoda crear clases y jerarquías, declarar propiedades para las clases, crear instancias e introducir valores. Todo ello gracias a una interfaz gráfica con menús, botones, cuadros de diálogo y representaciones gráficas muy intuitiva y fáciles de usar, como se puede observar en la figura 2.11.

Sus últimas versiones incluyen útiles herramientas de visualización gráfica de ontologías, control de versiones y fusión de ontologías.

Posee dos características destacables que están convirtiendo a esta herramienta en un entorno muy potente:

- Es un entorno abierto que se puede descargar libremente.
- Es fácil de extender, lo que permite que la comunidad contribuya activamente en la ampliación del entorno mediante la creación de plug-ins que están permitiendo su integración con un gran número de herramientas, aplicaciones, bases de conocimiento, y formatos de almacenamiento.

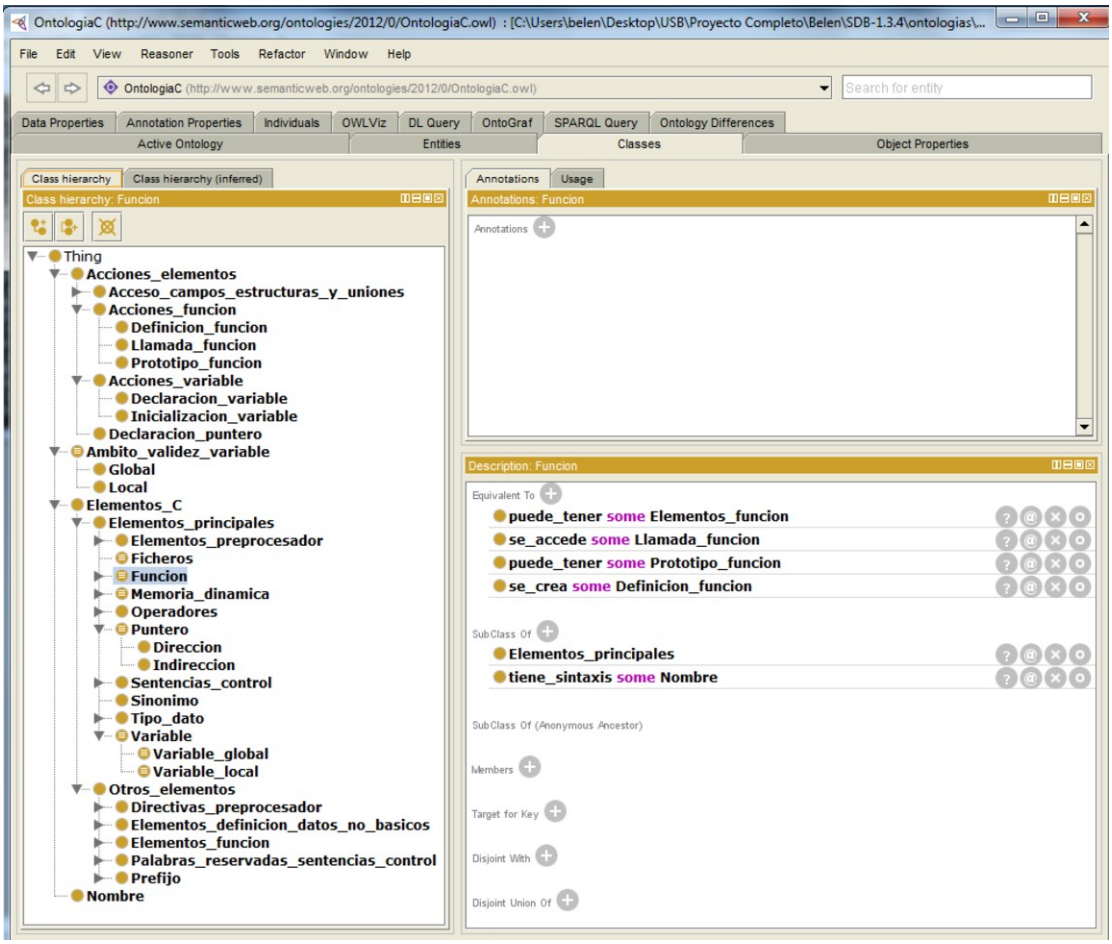


Figura 2.11: Entorno Protégé

3. ESPECIFICACIÓN DE REQUISITOS

Para construir un software de calidad, conviene realizar una especificación de requisitos de la aplicación previa a su desarrollo. En este capítulo se enumeran todos los requisitos, clasificados en funcionales y no funcionales, que el software debe cumplir para garantizar los objetivos del proyecto.

Cada requisito debe estar identificado por un nombre único e incluir una prioridad, para conocer su impacto sobre la funcionalidad completa, y una descripción clara y detallada.

Todos los requisitos son verificables, consistentes y compatibles. Además, han sido validados mediante pruebas funcionales en la que se han ido identificando problemas para su posterior corrección. Así sucesivamente hasta que los resultados obtenidos fueron los esperados.

3.1. Requisitos funcionales

1. **Ejecución de la aplicación**

   - Tipo: Funcional
   - Prioridad: Media.
   - Descripción:
     La aplicación que se debe desarrollar es una herramienta de ayuda, por lo que la decisión sobre si ejecutarla o no bebe ser del alumno y no del sistema.
     La ejecución se hará por tanto de forma manual por parte del alumno, con tan sólo introducir una sentencia corta y sencilla desde línea de comandos.

2. **Visualización de la aplicación**

   - Tipo: Funcional
   - Prioridad: Media.
   - Descripción:
     Hasta que la aplicación no detecte una primera compilación y obtenga datos con los que poder ofrecer algún tipo de ayuda, se ejecuta de forma pasiva (sin que se visualice nada) para no interferir en el trabajo habitual del alumno.

3. **Entrada de compilaciones**

   - Tipo: Funcional

- Prioridad: Alta.
- Descripción:

  La aplicación necesita una entrada de datos que indique las compilaciones que realiza el alumno y los errores obtenidos en cada una de ellas. Para ello ha de ser compatible y poder comunicarse con la herramienta 'gcc' que incluye la Maquina virtual.

  Dicha herramienta genera un archivo de texto llamado 'gcc.txt' donde se almacenan todos los errores de compilación producidos en el sistema.

4. **Comprobación constante de nuevas compilaciones**

   - Tipo: Funcional.
   - Prioridad: Media.
   - Descripción:

     Desde el momento en el que la aplicación detecta la primera compilación, se debe comprobar constantemente si el alumno genera una nueva compilación. De ser así se mostrará un aviso informando de esta situación.

5. **Mostrar los errores de compilación al usuario**

   - Tipo: Funcional.
   - Prioridad: Alta.
   - Descripción:

     La aplicación debe mostrar al alumno los errores ocurridos en su última compilación sobre los que se le puede ofrecer ayuda. Además se debe desarrollar una interfaz adecuada que dé la opción de seleccionar uno de ellos.

     Si no hay errores de este tipo, se mostrará el aviso correspondiente.

6. **Conexión con la base de datos SDB**

   - Tipo: Funcional.
   - Prioridad: Alta.
   - Descripción:

     La aplicación ha de ser capaz de conectarse con la base de datos SDB donde se almacenan todas las ontologías y anotaciones de los recursos educativos. Además debe poder realizar consultas SPARQL sobre los datos y obtener respuestas que actuaran como entrada de datos de la aplicación.

7. **Recorrer la ontología**

   - Tipo: Funcional.
   - Prioridad: Alta.

- Descripción:
  Se debe desarrollar un algoritmo capaz de cargar la ontología que se corresponda con el error seleccionado por el alumno, e ir recorriéndola hasta llegar a las clases que se encuentran más abajo en la jerarquía de clases y que se corresponden con las diferentes causas específicas para ese error.
  Para poder ir recorriendo la ontología e ir seleccionando el camino correcto, se basa en la interacción con el alumno que se especifica en el requisito 8, y en consultas SPARQL a la base de datos SDB que contiene las ontologías.

8. **Elección de alternativas**

   - Tipo: Funcional.
   - Prioridad: Alta.
   - Descripción:
     Cada vez que sea necesario, se deben mostrar al alumno las diferentes alternativas que surgen al recorrer la ontología. Además se debe desarrollar una interfaz adecuada que dé la opción de seleccionar una de ellas para que el algoritmo explicado en el requisito 7 conozca el camino a tomar y pueda proseguir recorriendo la ontología.

9. **Obtener lista de causas**

   - Tipo: Funcional.
   - Prioridad: Alta.
   - Descripción:
     Cuando el algoritmo explicado en el requisito 7 ha terminado de recorrer la ontología y tiene identificadas las clases que cumplen con todas las alternativas que el alumno ha ido seleccionando, se debe crear una lista con dichas clases que representan a las posibles causas del error.

10. **Obtener la causa del error**

    - Tipo: Funcional.
    - Prioridad: Alta.
    - Descripción:
      Una vez se dispone de la lista especificada en el requisito 9, se debe desarrollar una interfaz capaz de ir mostrando una a una cada una de estas posibilidades para que el alumno pueda identificar cuál de ellas es la que se corresponde con la causa de su error.

11. **Mostrar información y enlaces sobre la causa del error**

    - Tipo: Funcional.
    - Prioridad: Alta.
    - Descripción:
      Una vez identificada la causa del error, se debe mostrar al alumno:

- Conocimiento sobre ciertos conceptos relacionados con la causa y definidos en la ontología sobre el propio dominio de programación en C.
- Un listado de recursos web educativos que ayuden al alumno en la corrección y comprensión de la causa de sus propios errores.

12. **Distinción entre recursos educativos con teoría o ejercicios**

    - Tipo: Funcional.
    - Prioridad: Media.
    - Descripción:
      En el listado de recursos web explicado en el requisito 11, se deben clasificar clasificados en 'Teoría', con aquellos recursos que sólo ofrecen información teórica, y 'Ejercicios' con enlaces a recursos que ofrecen prácticas o ejercicios y no únicamente contenido teórico.
      El criterio para poder clasificar en 'Teoría' o 'Ejercicios' se basa en que los enlaces que contienen ejercicios deben estar etiquetados como tales.

13. **Ordenación de los recursos educativos**

    - Tipo: Funcional.
    - Prioridad: Media.
    - Descripción:
      Los recursos educativos deben aparecer colocados por orden de importancia de forma que el más relevante se muestre el primero y el menos importante el último.

14. **Interfaz funcional**

    - Tipo: Funcional.
    - Prioridad: Media.
    - Descripción:
      Para que la interfaz resulte funcional, en todo momento se debe facilitar la reversibilidad de las acciones y mostrar mensajes explicativos e informativos que indiquen al alumno las opciones que ya ha seleccionado anteriormente para llegar a ese punto.

3.2. Requisitos no funcionales

15. **Fiabilidad**

    - Tipo: No funcional. Rendimiento del sistema.
    - Prioridad: Alta.
    - Descripción:
      Toda la información que muestra la aplicación debe ser exacta y fiable, pues ha de servir de ayuda en el aprendizaje del alumno. No será permisible que se de información errónea.

16. **Disponibilidad**

    - Tipo: No funcional - Rendimiento del sistema.
    - Prioridad: Media.
    - Descripción:

      Una vez que la aplicación detecta una compilación y puede ofrecer ayuda, debe estar disponible el 100 % del tiempo hasta que el alumno proceda a su cierre.

17. **Tiempo de respuesta**

    - Tipo: No funcional - Rendimiento del sistema.
    - Prioridad: Alta.
    - Descripción:

      Para poder ofrecer una interacción adecuada, el tiempo de respuesta debe ser el adecuado para que el alumno lo interprete como casi inmediato y no tenga que esperar. Aunque la aplicación interactúa con otros sistemas, lo que dificulta un buen tiempo de respuesta, en todo caso dicho tiempo no puede ser superior a 3 segundos y sería ideal que fuera menos.

18. **Utilización de recursos**

    - Tipo: No funcional.
    - Prioridad: Alta.
    - Descripción:

      La utilización de recursos del sistema debe ser la menor que garantice un correcto funcionamiento de la aplicación.

      Existe un compromiso 'utilización de recursos-tiempo de respuesta' que debemos asumir. Para que el tiempo de espera sea pequeño la aplicación debe consumir más recursos, y si minimizamos el número de recursos provocamos un tiempo de respuesta elevado que podría hacer esperar demasiado al alumno.

19. **Capacidad**

    - Tipo: No funcional - Rendimiento del sistema.
    - Prioridad: Media.
    - Descripción:

      Para no saturar y evitar una posible sobrecarga de información tanto del sistema como del propio alumno, el número máximo de errores ocurridos en una misma compilación que deben ser tratados por la aplicación es 100. En caso de haber más, se usarán los cien primeros.

20. **Entorno**

    - Tipo: No funcional - Diseño.
    - Prioridad: Alta.

- Descripción:

  La aplicación se debe integrar dentro de una Máquina virtual con sistema operativo Ubuntu, concretamente la que usan los alumnos que cursan la asignatura 'Arquitectura de Sistemas'.

  Dicho entorno cuenta además con la herramienta 'gcc' que será necesaria para la elaboración de la aplicación.

21. **Lenguaje de programación**

- Tipo: No funcional - Diseño.
- Prioridad: Alta.
- Descripción:

  El lenguaje de programación que se debe utilizar es Java que nos aporta beneficios en seguridad, portabilidad y escalabilidad.

22. **Memoria en la base de datos**

- Tipo: No funcional.
- Prioridad: Alta.
- Descripción:

  Se debe disponer de una base de datos adecuada con memoria suficiente para almacenar todas las ontologías y anotaciones necesarias.

23. **Comunicación con la base de datos**

- Tipo: No funcional - Diseño.
- Prioridad: Alta.
- Descripción:

  Se debe disponer de un entorno Jena, necesario para poder leer, recorrer y modificar información en grafos tanto RDF como OWL desde un programa Java. Para el almacenamiento en la base de datos y desarrollar consultas a ella mediante SPARQL, se debe emplear un componente de Jena llamado SDB.

  Además se dispondrá del conector Java correspondiente a dicha base de datos.

24. **Apariencia de la interfaz**

- Tipo: No funcional - Interfaz.
- Prioridad: Media.
- Descripción:

  Se deben usar fuentes y tamaños de letra, colores y estructuraciones de texto parecidas a las empleadas en los recursos web de la asignatura 'Arquitectura de sistemas'. De esta forma los alumnos interpretaran la relación de la aplicación con el aprendizaje de la asignatura.

  La interfaz desarrollada debe tener una resolución lo suficientemente grande para poder mostrar de forma rápida y cómoda toda la información necesaria, pero sin

ocupar toda la resolución de la pantalla para que el alumno tenga espacio para navegar y consultar su código a la vez que interacciona con la aplicación. Un tamaño de resolución adecuado es 920x650.

25. **Idioma**

    - Tipo: No funcional - Interfaz.
    - Prioridad: Baja.
    - Descripción:
      El idioma por defecto en el que se debe mostrar la información es el español. Pero, ya que se dispone de los recursos educativos también en inglés, se debe dar la opción de poder obtener enlaces a páginas en inglés.

26. **Manejo de la interfaz**

    - Tipo: No funcional - Utilización.
    - Prioridad: Media.
    - Descripción:
      La interfaz de la aplicación debe ser muy intuitiva y sencilla de manejar para que el tiempo de aprendizaje para poder usarla de forma correcta sea mínimo.

27. **Mantenimiento**

    - Tipo: No funcional.
    - Prioridad: Media.
    - Descripción:
      La aplicación debe poderse adaptarse de forma sencilla a los nuevos cambios que puedan producirse en los recursos educativos. Por ello se deben identificar los cambios en el código y en las anotaciones que se han de realizar si esto ocurre para poder actualizar rápidamente a la vez que evolucionan los recursos web.

28. **Ayuda**

    - Tipo: No funcional.
    - Prioridad: Baja.
    - Descripción:
      El software debe incorporar una ayuda que incluya una descripción del funcionamiento de la interfaz de la propia aplicación, orientada a las acciones que debe realizar el alumno.

29. **Libre de copyright**

    - Tipo: No funcional.
    - Prioridad: Media.
    - Descripción:
      En ningún momento se puede mostrar nada que contenga copyright. Para evitar este problema, se deben elaborar iconos y botones propios.

4. DISEÑO E IMPLEMENTACIÓN

A lo largo de este capítulo se detalla el diseño y la implementación de la aplicación desarrollada en este proyecto.

En primer lugar, en la sección 4.1, se realiza una descripción de todas las ontologías, así como de los pasos seguidos para su creación.

A continuación, en la sección 4.2, se explica la anotación de los recursos educativos que se ha llevado a cabo y se presentan los nuevos recursos elaborados como apoyo de los existentes, para así poder devolver información útil sobre todas las causas de error definidas en las ontologías.

En 4.3 se explica la lógica del software desarrollado. Para ello se incluyen diagramas de flujo de la funcionalidad completa y de las tareas más importantes. Además se especifica la estructuración del código y se presenta un diagrama de clases. Por último se presentan los recursos web elaborados como apoyo de los existentes, para así poder devolver información útil sobre todas las causas de error definidas en las ontologías.

Todas las características de la interfaz gráfica de la aplicación y la forma de interacción con los alumnos, se detallan en la sección 4.4.

4.1. Descripción de ontologías

En este apartado se mostrarán y especificarán las ontologías desarrolladas, pero antes vamos a detallar su proceso de creación que comienza por identificar los diferentes elementos que la componen. Las partes en las que se divide una ontología son:

- Clase: descripción formal de una entidad del dominio que se quiere representar. Es la pieza básica de estructuración del conocimiento ya que representa los conceptos del dominio.

  Cada clase se identifica por un nombre que, aunque ha de estar relacionado, no tiene porque coincidir con la denominación del concepto al que representa.

  Existe la herencia de clases, por lo que una clase (a la que llamaremos clase padre) puede tener subclases (llamadas clases heredadas) que poseen todas las características de la clase de la que heredan más otras propias, por lo que es una forma de representar conceptos más específicos. También puede producirse herencia múltiple, que se da cuando una clase hereda de dos o más clases padre.

- Instancia: representan objetos concretos del dominio que pertenecen a una determinada clase.
- Propiedad: define las características propias de una clase y sus instancias y marca la interacción o enlaces entre clases del dominio que se modeliza.

  Los valores que puede tomar una propiedad están sometidos a restricciones como por ejemplo: tienen que ser un cardenal, debe estar dentro de un rango, o puede ser obligatoria o no.
- Axioma: regla que, aportando información adicional sobre el comportamiento de las clases, va a permitir hacer deducciones lógicas sobre ellas. Permiten dejar constancia de que ciertos valores de propiedades introducidos son coherentes con las restricciones de la ontología, o incluso deducir posteriormente valores que no se han introducido explícitamente.
- Anotaciones: texto que se incluye en instancias o clases para completar información. La forma más común de anotación consiste en añadir etiquetas semánticas con dicho texto.

El desarrollo de ontologías no es una tarea fácil, por lo que se ha necesitado realizar un profundo trabajo de investigación en el que se han consultado diversos trabajos anteriores, como [35], [36], [37], [16] y [38], y elaborado un análisis concreto de nuestras propias necesidades.

Partiendo de las referencias indicadas, se ha identificado que los principales pasos a seguir para el diseño y desarrollo de una ontología son los que se muestran a continuación. En cada uno de ellos se especifican las decisiones tomadas para la elaboración de las ontologías de este proyecto:

1. El primer paso requiere un arduo análisis ontológico del dominio del conocimiento que se va a representar, analizando el propósito y el alcance de la ontología.

   En este caso, el objetivo es la educación por lo que debe seguirse una estructura sencilla y no utilizar conceptos demasiados complejos, ya que los usuarios a los que va dirigidos no tienen un nivel alto de conocimiento sobre el dominio. Se deben lograr ontologías completas, descriptivas y sencillas.

2. El segundo paso es investigar si se han desarrollado ontologías previas con el mismo dominio de conocimiento.

   Esto es importante ya que si existen ontología iniciales de las que partir, tan sólo habría que modificar las existente para satisfacer las nuevas necesidades, lo que es mucho más sencillo que crear una nueva ontología desde cero. Además, la reutilización de ontologías existentes puede garantizar la compatibilidad con otros sistemas o aplicaciones que ya usaban esas ontologías particulares.

   En el caso concreto de las ontologías desarrolladas para este proyecto, la reutilización no ha sido posible, ya que no existen ontologías completas sobre el lenguaje de programación en C ni sobre sus errores de compilación, por lo que se decidió desarrollarlas desde cero y así garantizar que satisfacían el propósito tan específico que persigue este proyecto.

3. El tercer paso es enumerar e identificar los términos importantes en la ontología, que se transformaran en clases o instancias.

   Es útil hacer una lista con todos los términos que tienen relación con nuestro dominio, para luego identificar cuales se transformarán en clases, cuales en propiedades y cuales en instancias.

   Este fue el paso más importante en el desarrollo de las ontologías, ya que el dominio de conocimiento es muy complejo, con muchos conceptos relacionados entre sí, y que requiere de un alto conocimiento sobre el lenguaje de programación C. Se necesitó profundizar sobre el propio dominio y consultar mucha documentación.

4. Definir las clases y su jerarquía.

   Determinar qué elementos se definen como clases no es fácil, y debe decidirse en función de los objetivos de la ontología. Una de las decisiones más difíciles e importantes a la hora de modelizar ontologías es decidir cuándo crear o introducir una nueva clase intermedia. Para tomar esta decisión hay que tener en cuenta que las subclases presentan propiedades adicionales que su clase padre no posee, o tienen restricciones o bien participan en relaciones distintas. Debido a ello, introduciremos una nueva clase en la jerarquía cuando hay algo que podemos decir de ella que no podemos afirmar de su clase padre. En la práctica todas las subclase deben tener las propiedades de la clase de la que heredan más nuevas propiedades o nuevos valores para las propiedades.

   No existe una única jerarquía correcta para un dominio, depende de los posibles usos de la ontología, del nivel de detalle necesario para sus futuras aplicaciones, del enfoque personal del creador e incluso de la compatibilidad con otros sistemas con los que se va a utilizar.

   Para decidir e identificar las clases y las relaciones entre ellas, existen dos técnicas [38]:

   - Técnica top-down: comienza con la definición de los conceptos o clases más generales del dominio y posteriormente se lleva a cabo su especialización con la creación de clases más específicas.
   - Técnica botom-up: comienza con la definición de los conceptos o clases más específicas, que reagruparemos posteriormente en clases que definen conceptos más generales.

   Para la elaboración de las ontologías que figuran en este documento, se ha usado una combinación de las dos técnicas explicadas anteriormente. En primer lugar se realizó una enumeración de los conceptos más importantes del dominio que no podían faltar en la ontología, y finalmente se especializaron y generalizaron apropiadamente creando a partir de ellos nuevas subclases y clases más generales.

5. Una vez que se han definido las clases, hay que describir las propiedades que definen y relacionan los conceptos.

   Una propiedad debe ser adscrita a la clase más general que posea dicha propiedad. Para definirlas hay que especificar el tipo de valor asociado, valores permitidos, número de valores permitidos y todas las características que se consideren necesarias.

A la hora de modelizar un dominio, es difícil decidir si modelizar una distinción específica entre conceptos con un valor distinto de una propiedad o como un nuevo conjunto de clases. Esta decisión depende de los límites que hayamos impuesto al dominio de la ontología y de la importancia que esa distinción tenga dentro de nuestro propio dominio.

6. El último paso es crear instancias.

   Las instancias se pueden entender como ejemplos de elementos que cumplen las propiedades de una clase. En nuestras ontologías no se han creado instancias ya que se han elaborado ontologías que definen el dominio de forma general, no aplicadas a un código específico. Para adaptar nuestras ontologías a un programa concreto, se podría por ejemplo instanciar dentro de la clase variable, el nombre de una variable determinada que se declara en él.

   Con esta decisión se ha limitado en cierto modo el alcance, pero una ontología no tiene por que contener toda la información posible sobre el dominio, simplemente especializar o generalizar el conocimiento que necesitemos para la aplicación que se va a desarrollar.

A continuación se detallan todas las ontologías desarrolladas en este proyecto. Para una mejor comprensión se muestran de forma gráfica. Las clases aparecen en cajas marcadas por un círculo y su nombre siempre empieza por mayúsculas. Las propiedades se representan mediante flechas, su nombre está en minúsculas y cada una de ellas se representa mediante un color distinto como se muestra en una pequeña leyenda que acompaña a cada figura.

Algunas de estas propiedades están definidas como 'equivalent class'. Esto significa que si cualquier instancia o miembro satisface esta propiedad con un valor adecuado, es necesariamente un miembro de la clase definida por esa propiedad.

4.1.1. Ontología sobre el propio dominio de programación en C

Se trata de una ontología muy amplia que incluye todos los conceptos importantes del lenguaje de programación C que son incluidos en la asignatura 'Arquitectura de Sistemas' de segundo curso de varios grados relacionados con Ingeniería de Telecomunicación, y las relaciones entre ellos. Esta ontología servirá para reflejar las conexiones entre diferentes conceptos de la programación en C, y servirá de nexo de unión con las causas de los errores de compilación cometidos por los alumnos y la propia área de conocimiento.

Debido a su complejidad y enorme extensión, a continuación tan sólo se presentan las partes más importantes de la misma. En la figura 4.1 aparece un listado con las clases principales que conforman la ontología. En las figuras 4.2, 4.3 y 4.4 se representan las clases y propiedades que se relacionan con algunos de los conceptos a modo ilustrativo, concretamente al concepto de variable, puntero y función respectivamente.

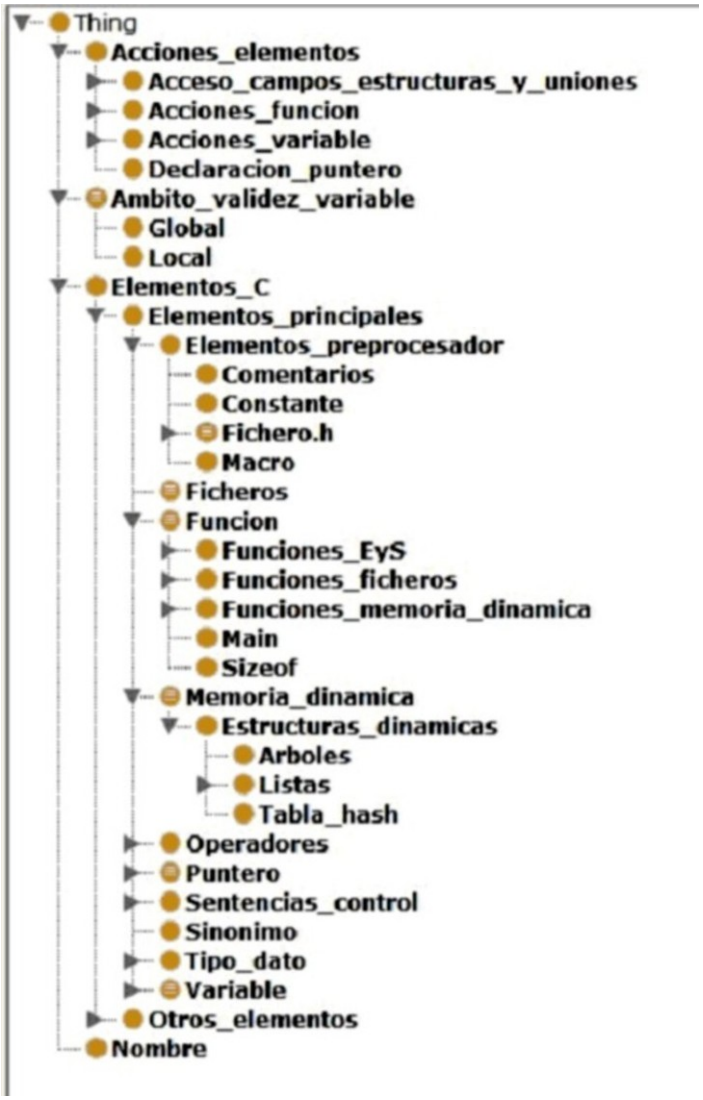


Figura 4.1: Jerarquía de las clases principales de la ontología sobre el lenguaje de programación C

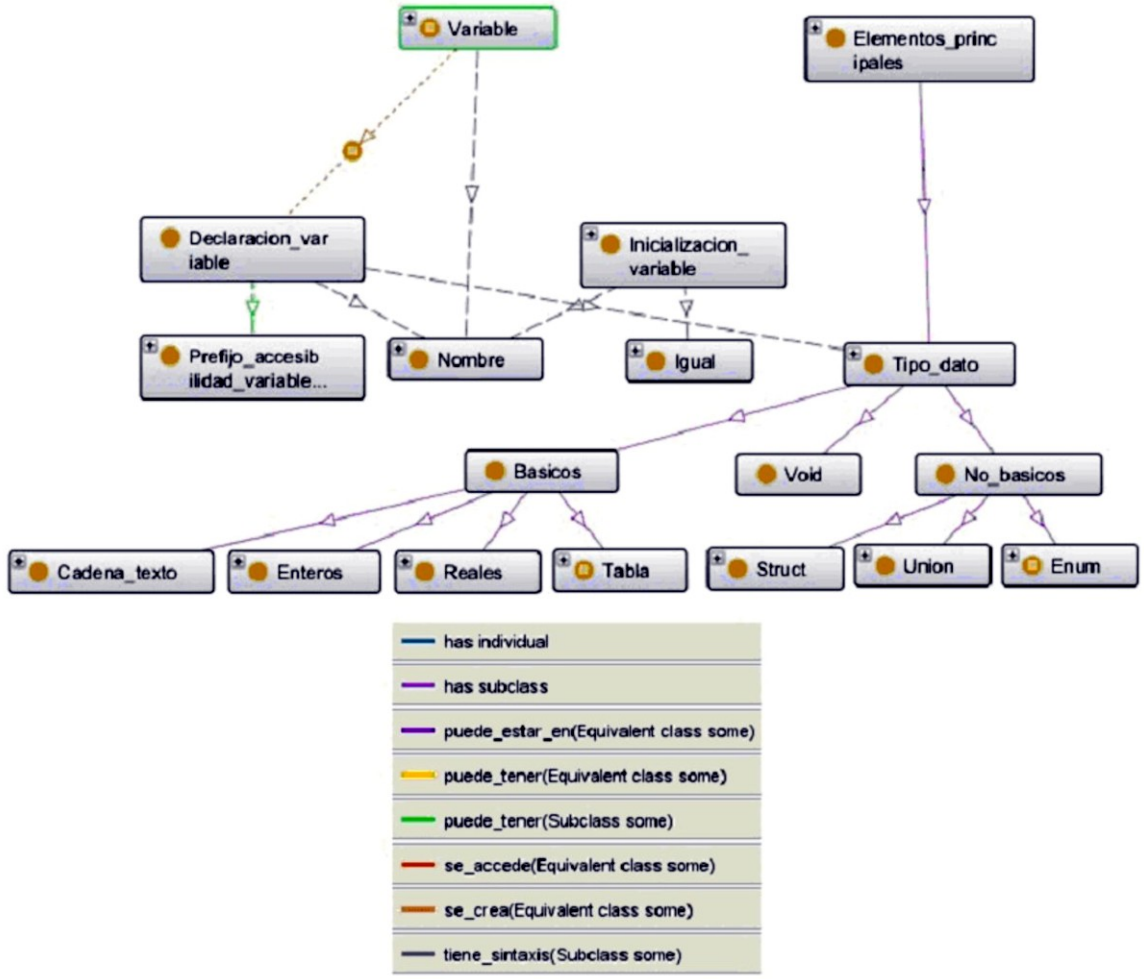


Figura 4.2: Parte de la ontología sobre el lenguaje de programación C que refleja el conocimiento sobre las variables

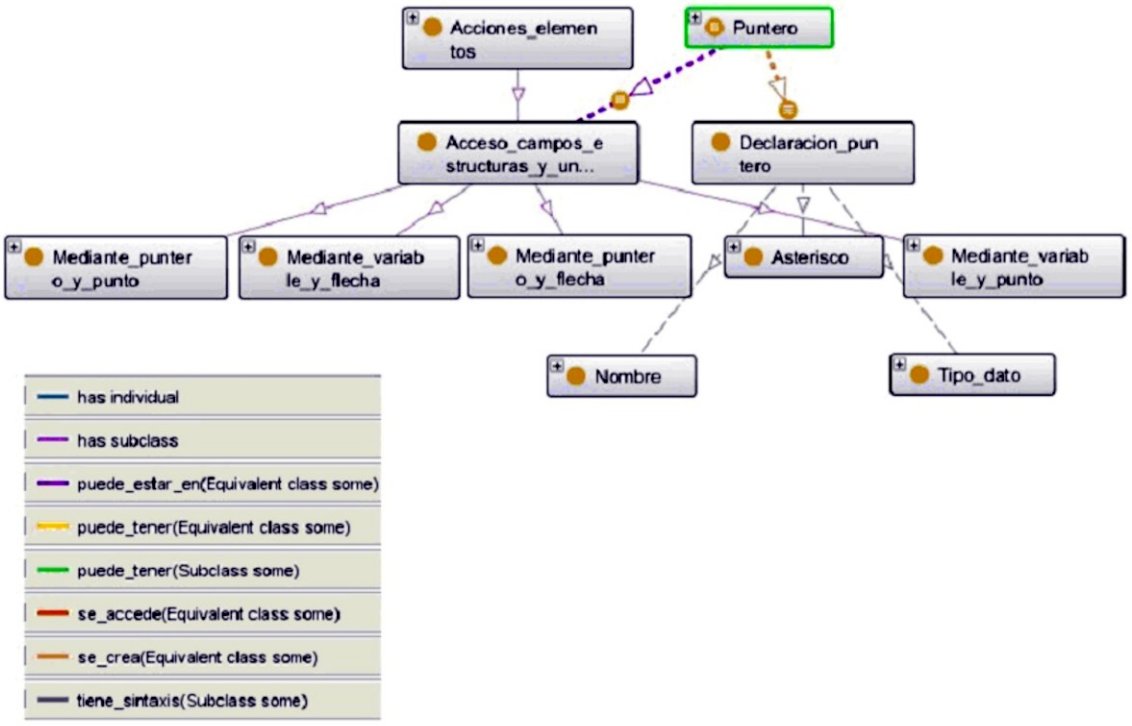


Figura 4.3: Parte de la ontología sobre el lenguaje de programación C que refleja el conocimiento sobre los punteros

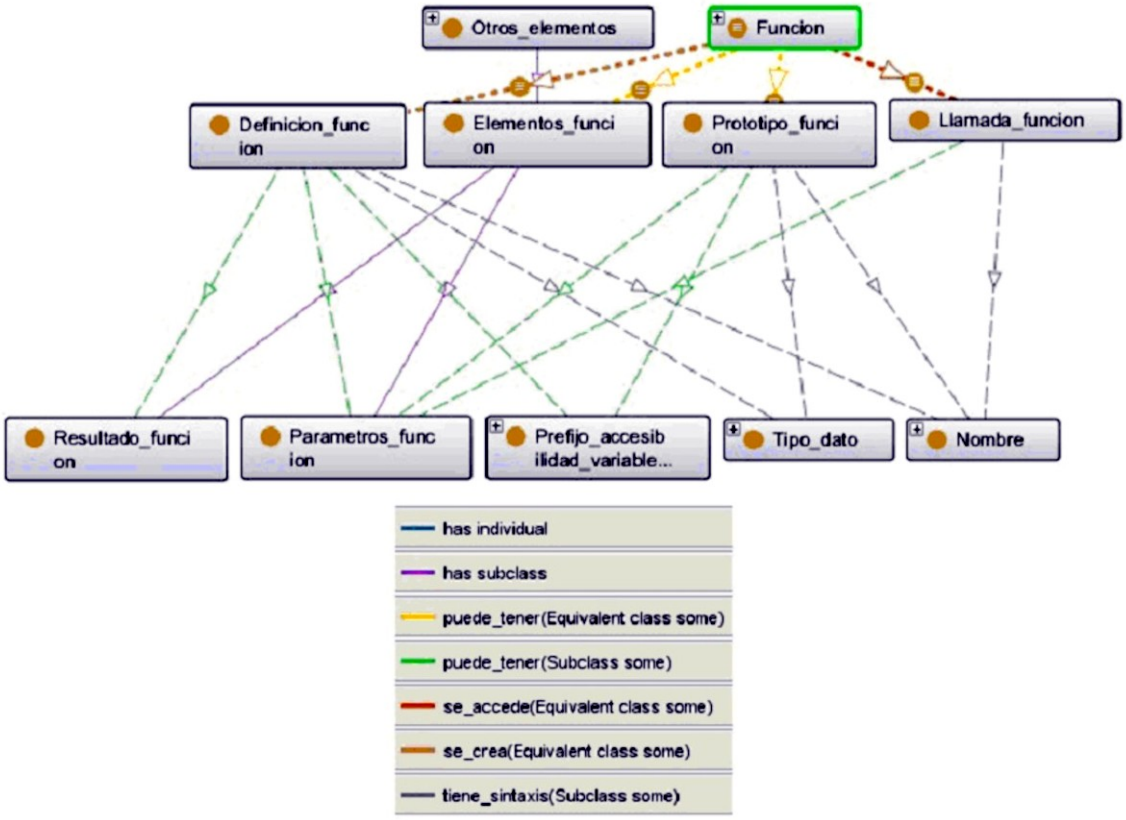


Figura 4.4: Parte de la ontología sobre el lenguaje de programación C que refleja el conocimiento sobre las funciones

4.1.2. Ontología sobre los errores de compilación

La ontología que se refleja en la figura 4.5, clasifica y define los 12 errores de compilación que trata la aplicación. Estos errores están entre los más comunes que los alumnos cometen mientras programan, tal como se muestra en [39].

Los 12 errores de compilación en los que se centra este proyecto y a los que se refieren los siguientes nombres de clases que aparecen en la ontología son:

1. **C99_mode**. Se refiere al error de compilación: 'for' loop initial declarations are only allowed in C99 mode

2. **Invalid_type_argument**. Se refiere al error de compilación: invalid type argument of '->'

3. **Deferencing_pointer**. Se refiere al error de compilación: dereferencing pointer to incomplete type

4. **Request_member**. Se refiere al error de compilación: request for member 'name_member' in something not a structure or union

5. **Redefinition_function**. Se refiere al error de compilación: redefinition of 'name_function'

6. **Few_arguments**. Se refiere al error de compilación: too few arguments to function 'name_function'
7. **Several_datatypes**. Se refiere al error de compilación: two or more data types in declaration specifiers
8. **Variable_undeclared**. Se refiere al error de compilación: 'name_variable' undeclared
9. **Conflicting_types**. Se refiere al error de compilación: conflicting types for 'name'
10. **Expected_statement**. Se refiere al error de compilación: expected statement before ...
11. **Expected_)**. Se refiere al error de compilación: expected ')' before ...
12. **Expected_;**. Se refiere al error de compilación: expected ';' before ...

Partiendo de este dominio de conocimiento, se ha desarrollado una nueva ontología para cada uno de los errores de compilación que se recogen en la ontología anterior. Estas ontologías tienen la finalidad de reflejar todas las posibles causas que provocan cada uno de los errores. A continuación explicaremos de forma más detallada las ontologías de cada uno de ellos, teniendo en cuenta que su jerarquía de clases se ha elaborado teniendo en cuenta su finalidad y por tanto se centran en especificar causas de error.

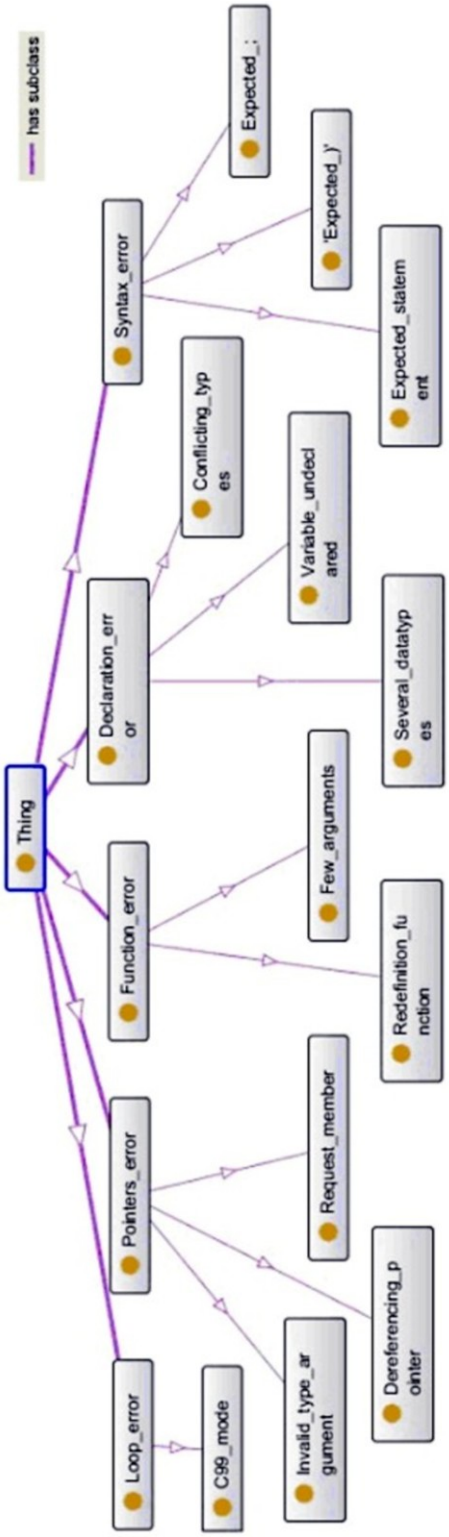


Figura 4.5: Ontología sobre los errores de compilación

4.1.3. Ontología sobre el error C99_mode

En la figura 4.6 se observa que sólo se ha identificado una causa que provoque este error y que está representada por la clase **Crear_variable_control_en_for**. Este nombre se refiere a que la variable de control usada en el bucle for no se puede declarar dentro del bucle.

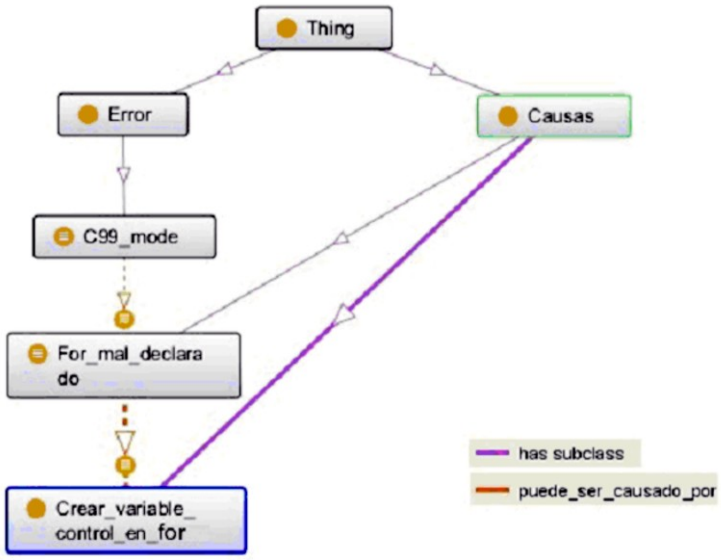


Figura 4.6: Ontología sobre el error C99_mode

4.1.4. Ontología sobre el error Invalid_type_argument

En la figura 4.7 se observan las siguientes causas específicas que pueden provocar este error. Como causas específicas consideramos a las clases que representan causas y que se encuentran en lo más bajo de la jerarquía de clases.

- **Necesita_operador_punto**: Entre el nombre de la variable de tipo estructura o unión y el nombre de uno de los campo de dicha estructura o unión no aparece el operador '.'.
- **Tipo_puntero_modificado**: El nombre que va precedido al operador ->se corresponde con el de un puntero pero va precedido de un operador '*' que modifica su tipo.
- **Faltan_parentesis**: El nombre que va precedido al operador ->se corresponde con el de una variable de tipo estructura o unión, va precedido de un operador & pero ambas cosas no aparecen entre paréntesis.
- **Debe_ser_puntero**: El nombre que va precedido al operador ->no es un puntero a una estructura o unión.
- **Falta_operador_ampersand**: El nombre que va precedido al operador ->se corresponde con el de una variable de tipo estructura o unión pero no va precedido de un operador &.

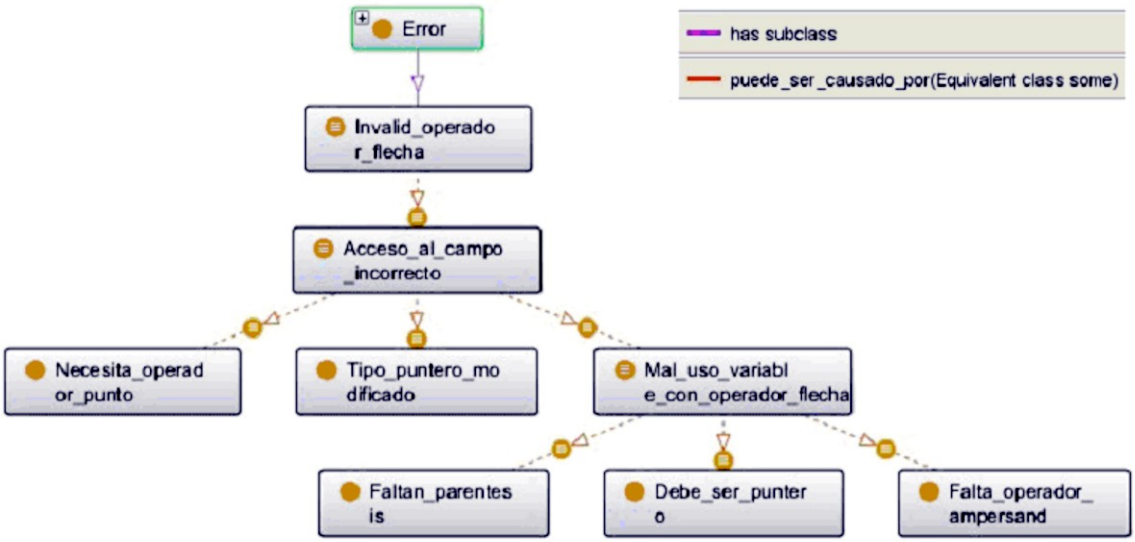


Figura 4.7: Ontología sobre el error Invalid_type_argument

Todas las causas anteriormente explicadas se pueden resumir en que para realizar el acceso a un campo a través de una variable, no se ha cumplido alguno de estos dos formatos válidos:

(&nombre_variable)->nombre_campo
nombre_variable.nombre_campo

4.1.5. Ontología sobre el error Deferencing_pointer

En la figura 4.8 se observan las siguientes causas específicas que pueden provocar este error:

- **Nombre_estructura_incorrecto**: El nombre usado en la definición de la estructura o unión no coincide con el nombre de la estructura o unión empleado para definir el tipo en la declaración del puntero.
- **Numero_palabras_incorrecto**: La definición de la estructura o unión es correcta. Falta algún elemento o la sintaxis no es correcta.
- **Falta_punto_y_coma**: Al final de la definición de la estructura o unión (o en la línea anterior a ella) que da tipo al puntero, no se ha añadido un ; y por lo tanto esta no es correcta.
- **Falta_definicion_estructura**: No existe una definición de la estructura o unión que da tipo al puntero ni en ese fichero.c, ni en un fichero.h que se incluya en él ni en ningún otro archivo que forme parte del mismo ejecutable.
- **Definida_despues**: Se está haciendo referencia al campo de una estructura o unión que no ha sido definida previamente, sino en líneas posteriores.

- **Ambito_validez_incorrecto**: El puntero se está usando y/o declarando fuera del rango de validez de la estructura que da tipo al puntero.
- **Incluir_fichero_.h**: La definición de la estructura o unión que da tipo al puntero se encuentra en un fichero.h que no ha sido añadido correctamente.
- **Necesita_ser_extern**: La definición de la estructura o unión se encuentra en otro fichero.c y no se ha añadido el prefijo extern a dicha estructura o unión.

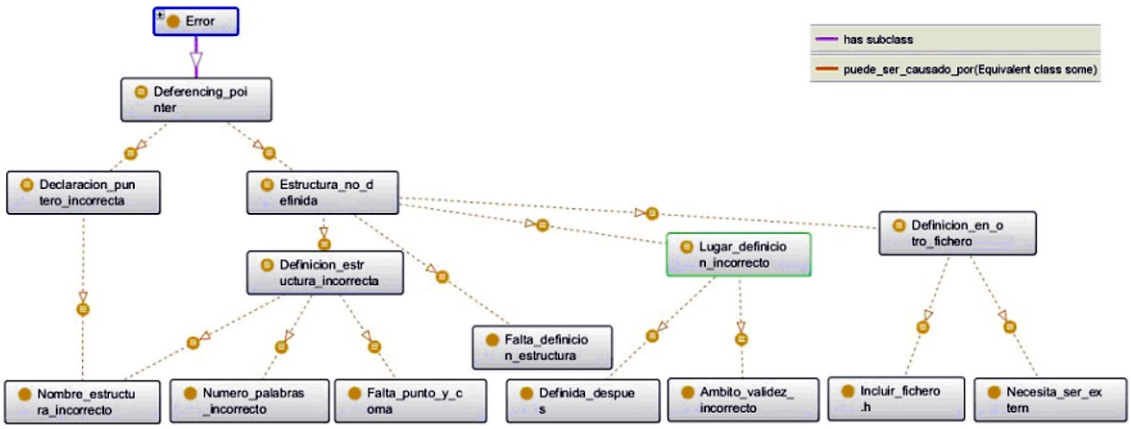


Figura 4.8: Ontología sobre el error Deferencing_pointer

4.1.6. Ontología sobre el error Request_member

En la figura 4.9 se observan las siguientes causas específicas que pueden provocar este error:

- **Necesita_operador_flecha**: Entre el nombre del puntero a una estructura o a una unión y el nombre de uno de los campo de dicha estructura o unión no aparece el operador ->.
- **Falta_operador_asterisco**: El nombre que va precedido al operador '.' se corresponde con el de un puntero a una estructura o unión pero no va precedido de un operador '*'.
- **Debe_ser_variable**: El nombre que va precedido al operador '.' no se corresponde con el de una variable cuyo tipo de dato es una estructura o unión.
- **Falta_parentesis**: El nombre que precede al operador '.' se corresponde con el de un puntero a una estructura o unión precedido de un operador '*', pero ambas cosas no están entre paréntesis.
- **Tipo_variable_modificado**: El nombre que va precedido al operador '.' se corresponde con el de un puntero pero va precedido de un operador '&' que modifica su tipo.

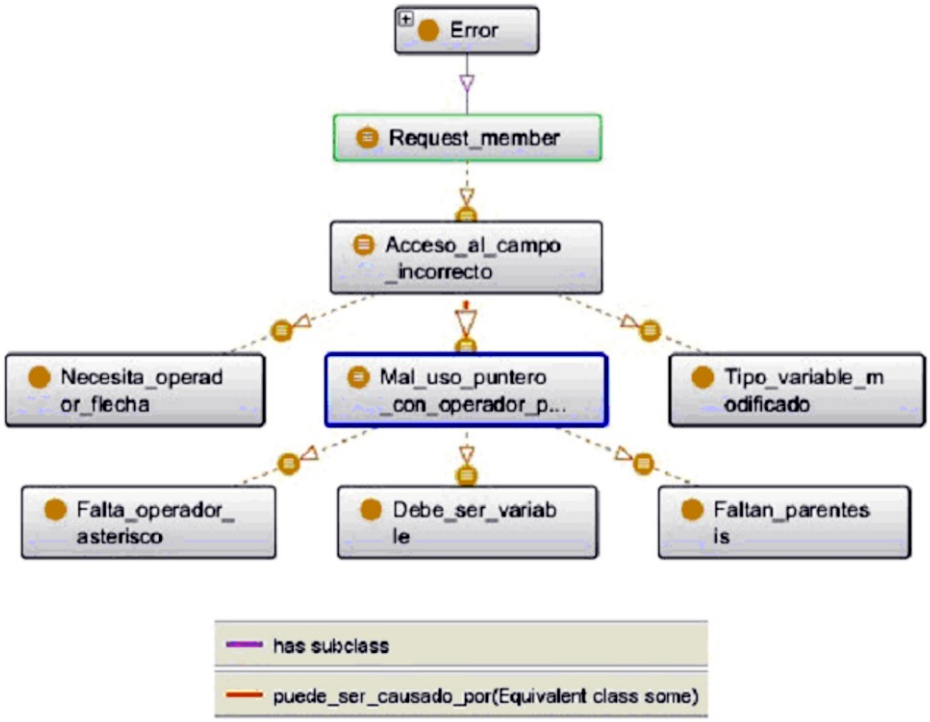


Figura 4.9: Ontología sobre el error Request_member

Todas las causas anteriormente explicadas se pueden resumir en que para realizar el acceso a un campo a través de un puntero, no se ha cumplido alguno de estos dos formatos válidos:

nombre_puntero->nombre_campo
(*nombre_puntero).nombre_campo

4.1.7. Ontología sobre el error Redefinition_function

En la figura 4.10 se observan las siguientes causas específicas que pueden provocar este error:

- **Redefinida_en_fichero.h**: Existe una definición de la función en un fichero.c y otra definición en algún fichero.h que se incluye a él.
- **Redefinida_en_mismo_fichero**: Existe más de una definición de la función en el mismo fichero.c.
- **Redefinida_en_varios_fichero.h**: Existe más de una definición de la función entre todos los fichero.h incluidos al fichero.c donde se produce el error.
- **Nombre_funcion_incorrecto**: El nombre de la función tiene una sintaxis incorrecta y coincide con el de otra función.

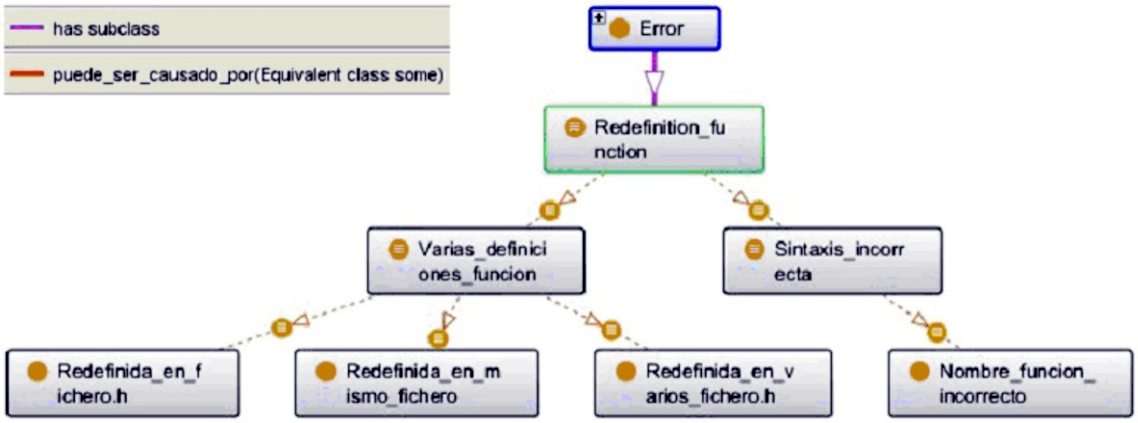


Figura 4.10: Ontología sobre el error Redefinition_function

4.1.8. Ontología sobre el error Few_arguments

En la figura 4.11 se observan las siguientes causas específicas que pueden provocar este error:

- **Numero_argumentos_incorrecto**: El número de argumentos de la definición de la función no coincide con el de la llamada a dicha función.
- **Nombre_funcion_incorrecto**: El nombre de la función tiene una sintaxis incorrecta y coincide con el de otra función.

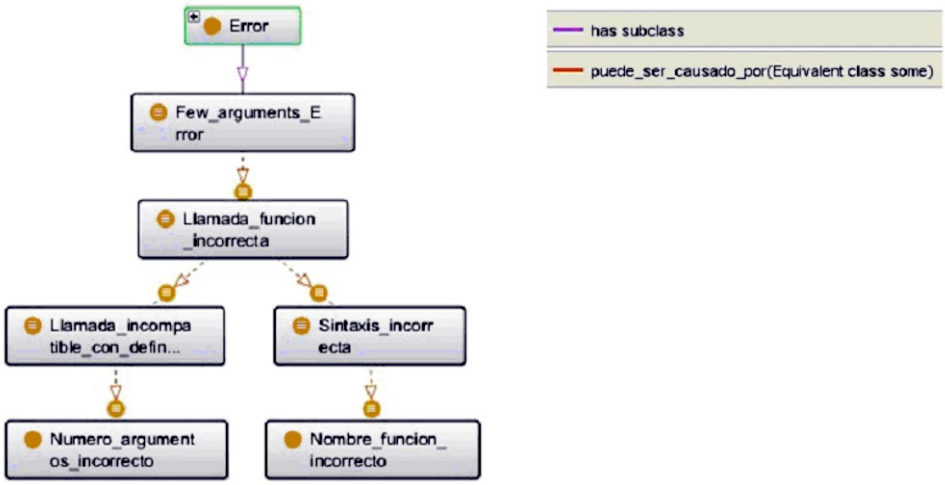


Figura 4.11: Ontología sobre el error Few_arguments

4.1.9. Ontología sobre el error Several_datatypes

En la figura 4.12 se observan las siguientes causas específicas que pueden provocar este error:

- **Demasiados_tipos_en_funcion**: En el prototipo o en la definición de una función se han asignado más de un tipo de datos de retorno.
- **Demasiados_tipos_en_parametros**: A algún parámetro del prototipo o la definición de una función, se le ha asignado más de un tipo de datos.
- **Demasiados_tipos_en_declaracion**: En la declaración de una variable se ha asignado más de un tipo de datos.

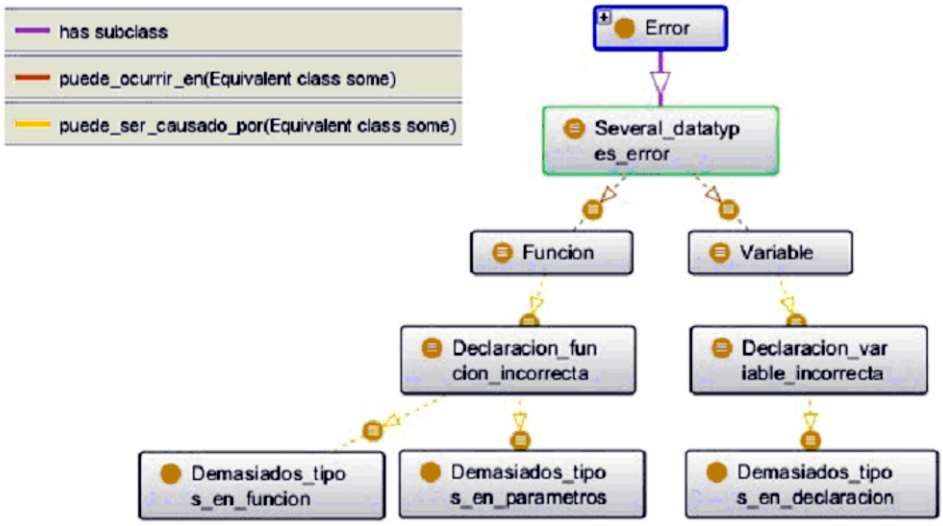


Figura 4.12: Ontología sobre el error Several_datatypes

4.1.10. Ontología sobre el error Variable_undeclared

En la figura 4.13 se observan las siguientes causas específicas que pueden provocar este error:

- **Definida_despues**: Se está accediendo a la variable antes de su declaración.
- **Ambito_validez_incorrecto**: Se esta intentando acceder a la variable fuera de su rango de validez.
- **Incluir_libreria**: No se ha añadido correctamente alguna librería, como stdio.h, que declara la variable.
- **Incluir_fichero.h**: La declaración de la variable se encuentra en un fichero.h y no se ha añadido correctamente este fichero.
- **Necesita_ser_extern**: La declaración de la variable se encuentra en otro fichero.c y no se ha añadido el prefijo extern a la variable.
- **Falta_punto_y_coma**: Al final de la declaración de la variable (o en la línea anterior a ella) no se ha añadido un ';' y por lo que la declaración no es correcta y el compilador no la ha reconocido como tal.

- **Numero_palabras_incorrecto**: La declaración de la variable no es correcta por lo que el compilador no la reconoce como tal.
- **Nombre_variable_incorrecto**: El nombre de la variable no coincide con el que aparece en la declaración. Existe un problema de sintaxis.
- **No_hay_declaracion**: No existe una declaración de la variable ni en el fichero donde se produce el error, ni en un fichero.h que se incluya en él ni en ningún otro archivo que forme parte del mismo ejecutable.

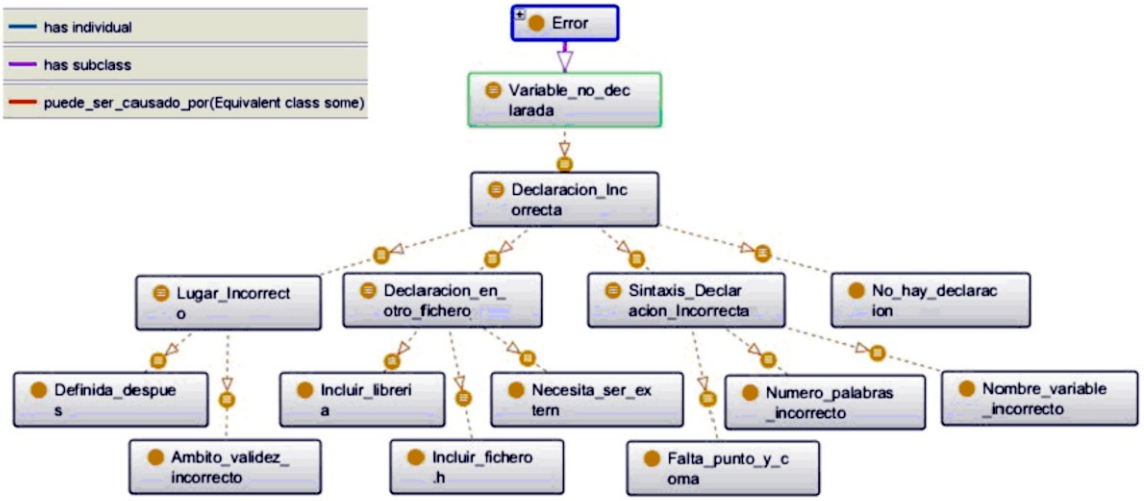


Figura 4.13: Ontología sobre el error Variable_undeclared

4.1.11. Ontología sobre el error Conflicting_types

En la figura 4.14 se observan las siguientes causas específicas que pueden provocar este error:

- **Doble_declaracion_variable**: Existe más de una declaración de la variable. Esto podría ocurrir tanto si aparecen dos declaraciones en el mismo fichero ó una en un fichero.c y otra en un fichero.h que se incluya a él.
- **Distinto_numero_argumentos**: El número de argumentos declarados en el prototipo de la función no coincide con el de la definición de dicha función.
- **Diferente_tipo_funcion**: El tipo de la función declarado en su prototipo no coincide con el de la definición de dicha función.
- **Diferente_tipo_argumentos**: El tipo de los argumentos declarados en el prototipo de la función no coincide con los de la definición de dicha función.
- **Doble_prototipo_funcion**: Existe más de un prototipo de la función con el mismo nombre pero diferente tipo o número de parámetros.

  Esto podría ocurrir tanto si aparecen dos declaraciones en el mismo fichero ó una en un fichero.c y otra en un fichero.h que se incluya a él.

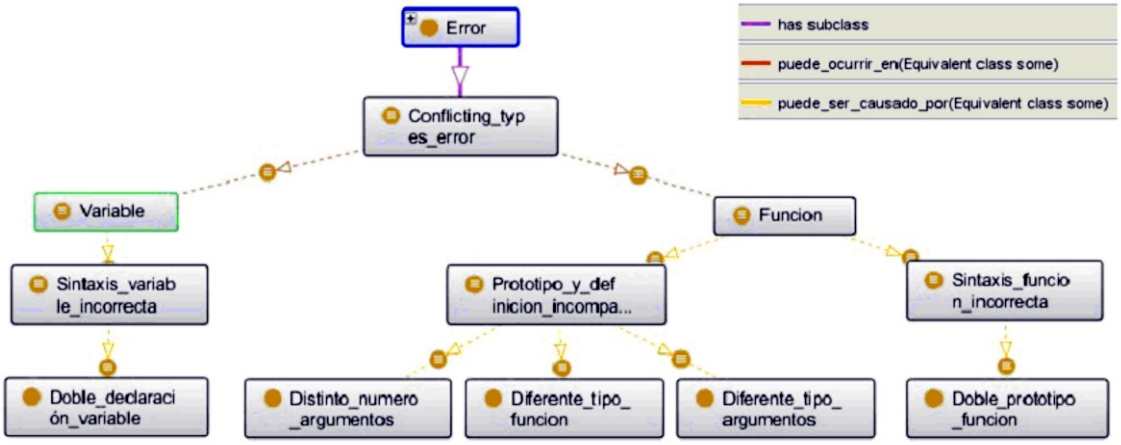


Figura 4.14: Ontología sobre el error Conflicting_types

4.1.12. Ontología sobre el error Expected_statement

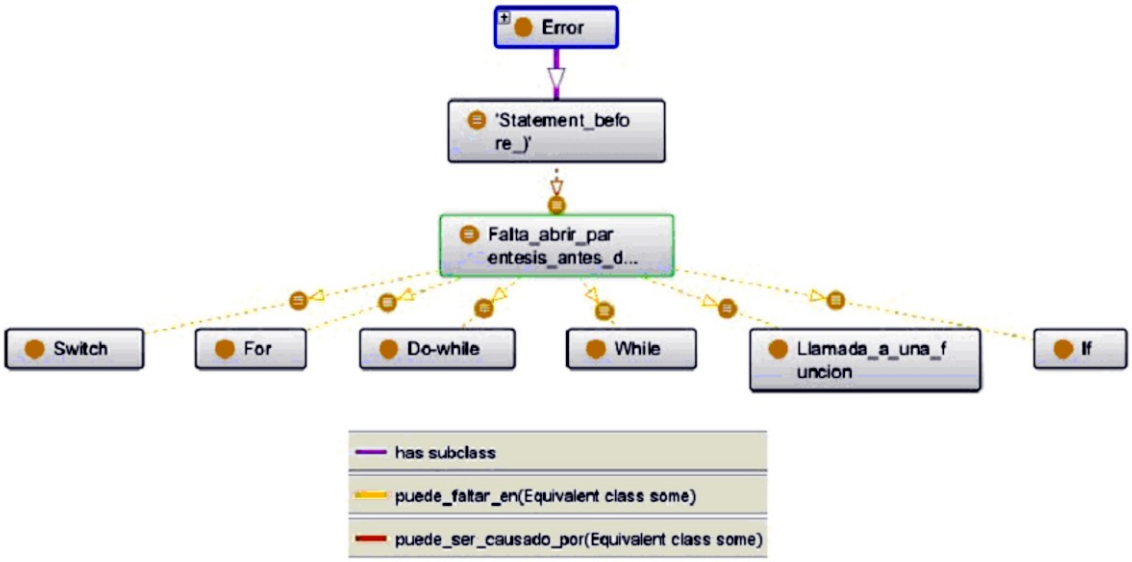


Figura 4.15: Ontología sobre el error Expected_statement

En la figura 4.15 se observan las siguientes causas específicas que pueden provocar este error:

- **Switch**: No se ha abierto paréntesis antes de poner la expresión que controla la ejecución del switch.
- **For**: No se ha abierto paréntesis antes de poner la condición del bucle for.
- **While**: No se ha abierto paréntesis antes de poner la condición del bucle while.
- **Do-while**: No se ha abierto paréntesis antes de poner la condición del bucle do-while.

- **Llamada a una funcion**: No se ha abierto paréntesis antes de indicar los valores que se pasan como parámetros en la llamada a una función.
- **If**: No se ha abierto paréntesis antes de poner la condición del bucle if.

4.1.13. Ontología sobre el error Expected_):

En la figura 4.16 se observan las siguientes causas específicas que pueden provocar este error:

- **If**: No se ha cerrado paréntesis tras poner la condición del bucle if.
- **Llamada_a_una_funcion**: No se ha cerrado paréntesis tras indicar los valores que se pasan como parámetros en la llamada a una función.
- **Switch**: No se ha cerrado paréntesis tras poner la expresión que controla la ejecución del switch.
- **Do-while**: No se ha cerrado paréntesis tras poner la condición del bucle do-while.
- **For**: No se ha cerrado paréntesis tras poner la condición del bucle for.
- **While**: No se ha cerrado paréntesis tras poner la condición del bucle while.

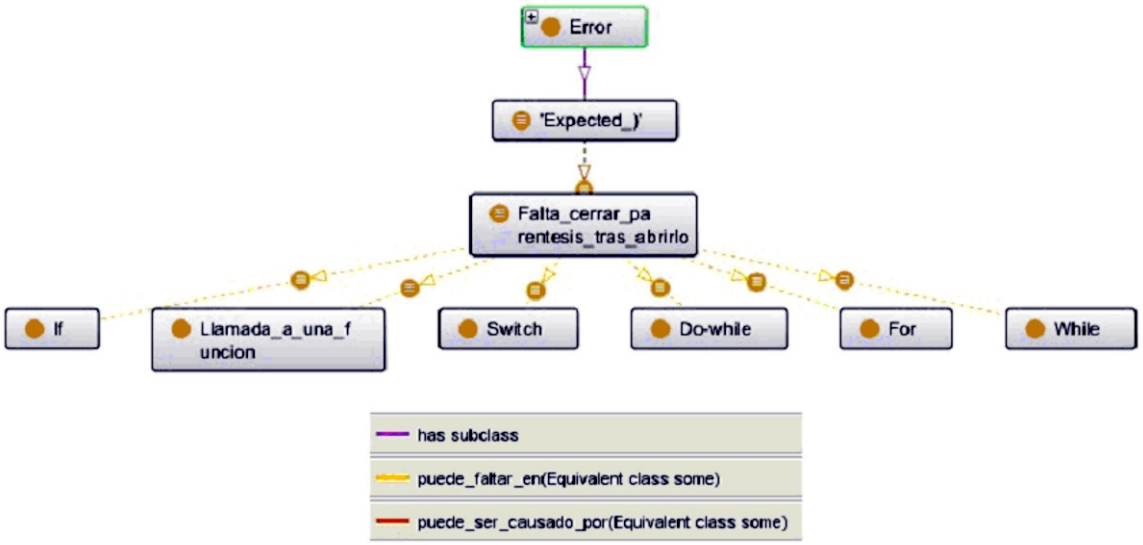


Figura 4.16: Ontología sobre el error Expected_):

4.1.14. Ontología sobre el error Expected_;

En la figura 4.17 se observan las siguientes causas específicas que pueden provocar este error:

- **Sentencia_anterior_incorrecta**: La sintaxis, en la línea donde se ha producido el error o en anteriores, no es correcta.

  Este error suele aparecer cuando alguna línea de código presenta algún problema de sintaxis por lo que el compilador interpreta algo que no es y espera encontrar un ; que no está.

- **Falta_;_en_sentencia_simple**: Hay una sentencia simple(declaraciones, asignaciones, llamadas a funciones, prototipos, sentencias aritméticas, etc) que no finaliza en un ;.

- **Falta_;_en_final_do-while**: No se ha añadido un ; al final de la declaración del bucle do-while.

- **Falta_;_en_declaracion_for**: No se ha añadido un ; entre la inicialización, la condición y/o la actualización del bucle for.

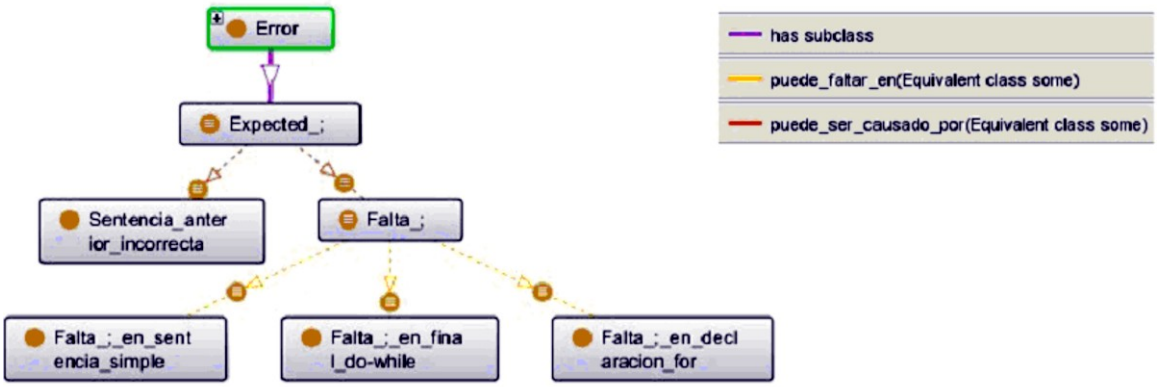


Figura 4.17: Ontología sobre el error Expected_;

4.2. Anotación de los recursos educativos

Una de las finalidades de esta aplicación es recomendar a los alumnos recursos web de la asignatura 'Arquitectura de Sistemas' relacionados con las causas de sus errores de compilación. Para saber con qué conceptos definidos en las ontologías se relaciona cada uno de estos recursos educativos, es necesario realizar una anotación RDF.

Esta anotación se declara en un archivo llamado 'Etiquetado.rdf'. Parte del contenido de este fichero se puede ver en la figura 4.18. En las primeras líneas del fichero se definen los prefijos de cada espacio de nombres, y a continuación se declara un 'rdf:Description' por cada uno de los recursos que deseamos anotar o etiquetar. El atributo 'rdf:about' identifica mediante una URL a cada recurso (o sujeto de la tripleta). El etiquetado se ha hecho a nivel de sección, por lo que las URLs pueden incluir un signo de almohadilla (#) seguido del identificador de fragmento para dirigir el navegador a un punto específico del recurso web, siempre que esta opción sea posible para esa página en concreto.

La propiedad que vamos a usar en las anotaciones se denomina 'label' y sirve para incluir cualquier texto. Su valor indica el tema principal del contenido del recurso. Para que tenga sentido en nuestra aplicación, este valor ha de corresponderse con el nombre de uno de los concepto que aparecen en la ontología sobre el lenguaje de programación C. Cuanto más específico sea este concepto; es decir, más abajo se encuentre dentro de la jerarquía de clases de la ontología, la aplicación será capaz de devolver recomendaciones más concretas sobre la causa del error de compilación.

La estructura de etiquetas para definir tripletas que usa RDF permite definir más de un valor para una determinada propiedad, por lo que se puede añadir más de un concepto para un mismo recurso.

Por necesidades de la aplicación, se debe distinguir entre recursos que contienen actividades, ejercicios o prácticas y recursos que incluyen únicamente información (teoría). Para indicar que una determinada URL hace referencia a un recurso que contiene ejercicios, hay que añadir al siguiente sentencia:

<dc:label>"Ejercicio" </dc:label>

Si el contenido fuera meramente teórico no se incluiría esta etiqueta. Por comodidad y sencillez, también se ha usado la propiedad 'label' ya que no deja de ser información sobre el propio contenido del recurso.

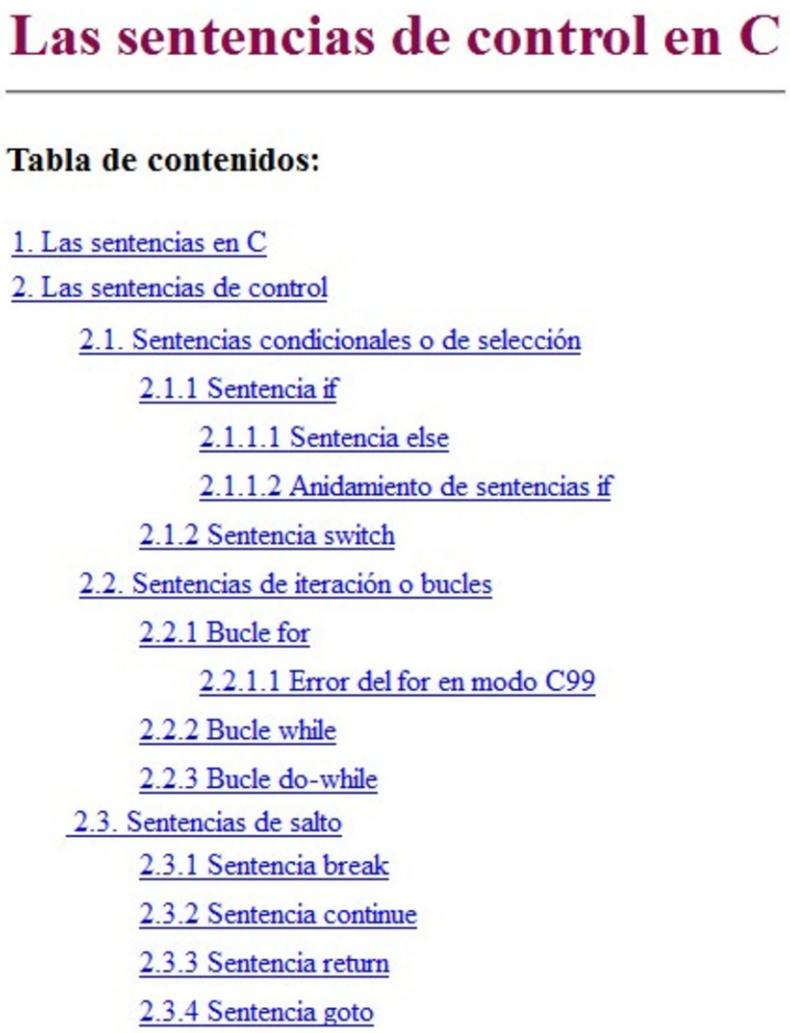
```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">

      <rdf:Description
rdf:about="http://www.it.uc3m.es/labas/course_notes/data_types_es.
html">
            <dc:label>Tipo_dato</dc:label>
      </rdf:Description>

      <rdf:Description
rdf:about="http://www.it.uc3m.es/labas/course_notes/variables_es.h
tml#variables_global">
            <dc:label>Variable_global</dc:label>
            <dc:label>Ambito_validez_variable</dc:label>
      </rdf:Description>

      <rdf:Description
rdf:about="http://www.it.uc3m.es/labas/course_notes/variables_es.h
tml#variables_static">
            <dc:label>Variable_local</dc:label>
            <dc:label>Ambito_validez_variable</dc:label>
            <dc:label>Extern</dc:label>
      </rdf:Description>

      <rdf:Description
rdf:about="http://www.it.uc3m.es/labas/course_notes/variables_acti
vities_structures_for_call_rates_es.html">
            <dc:label>Struct</dc:label>
            <dc:label>Ejercicio</dc:label>
      </rdf:Description>

      <rdf:Description
rdf:about="http://www.it.uc3m.es/labas/course_notes/functions_es.h
tml#functions_definition">
            <dc:label>Funcion</dc:label>
            <dc:label>Definicion_funcion</dc:label>
            <dc:label>Parametros_funcion</dc:label>
            <dc:label>Llamada_funcion</dc:label>
      </rdf:Description>

</rdf:RDF>
```

Figura 4.18: Fragmento de ’Etiquetado.rdf’ con la anotación de los recursos educativos

4.2.1. Desarrollo de nuevos recursos educativos

Para completar la funcionalidad de la aplicación y poder devolver información útil sobre todas las causas que se recogen en las ontologías desarrolladas, durante el desarrollo de este proyecto se han identificado aquellas causas que no se pueden relacionar con ningún recurso actual de la asignatura 'Arquitectura de Sistema' porque no se ofrece información sobre ellas (como por ejemplo todos los errores y causas relacionados con bucles), y se han elaborados nuevos recursos que ofrecen información sobre dichas causas no cubiertas. Estos nuevos recursos se encuentran actualmente almacenados en local en la ruta `$HOME/tutor/src/html`, aunque se podrían subir a un servidor ya que son de elaboración propia y no generarían ningún problema de copyright.

Concretamente se ha creado un fichero .html bajo el título 'Las sentencias de control en C'. La estética de este nuevo recurso sigue una organización, diseño, fuente y colores similares a los de la asignatura 'Arquitectura de Sistemas', para que no resulte a los alumnos demasiado diferente a lo que están acostumbrados a usar y su contenido les parezca fiable y fácil de consultar. La tabla de contenidos de este .html se puede ver en la figura 4.19.



Las sentencias de control en C

Tabla de contenidos:

Figura 4.19: Tabla de contenidos del recurso propio 'Las sentencias de control en C'

4.3. Lógica de la aplicación

En este apartado se explicará de forma detallada la lógica de todo el software desarrollado en este proyecto.

Para que la aplicación sea capaz de generar un feedback útil, se necesita realimentación. Esta realimentación se genera en primer lugar de forma implícita, ya que se recogen datos de las acciones del usuario de forma transparente para él. Se comprueba constantemente si el alumno ha generado una compilación y, de ser así, se obtiene una lista de errores ocurridos. Todo ello sin que el alumno deba realizar ninguna acción extra.

En segundo lugar, la realimentación se produce de forma explicita pidiendo información al usuario de forma directa mediante interacciones con él de manera que se pueda determinar las causas.

Explicar el diagrama de flujo completo de la aplicación resulta muy complejo, pues el control recae sobre el propio alumno que en cualquier momento puede volver atrás, volver a compilar, volver a consultar todos sus errores de compilación, consultar teoría, consultar la ayuda sobre el funcionamiento de la propia aplicación...

En la figura 4.20, se muestra como sería un diagrama de flujos completo desde que se ejecuta la aplicación hasta que el alumno consulta un recurso que le pueda ayudar a corregir y entender su error; es decir, el diagrama de flujos de la funcionalidad principal de la aplicación. En él se identifican las tareas que irán siendo detallas a continuación. Es el caso más sencillo, en el que el usuario ha obtenido errores de compilación sobre los que se le puede ofrecer ayuda y tan sólo interacciona respondiendo a la información que se le solicita (en el diagrama de flujo las respuestas del usuario se representan por líneas discontinuas). El resto de situaciones y acciones que el alumno puede realizar no se contemplan en este diagrama, aunque serán detalladas al final de esta sección.

Al ejecutar la aplicación, se inicia una tarea a la que llamaremos a partir de ahora 'Comprobar Compilación'. Esta tarea consiste en comprobar de forma recursiva si el usuario ha generado alguna compilación desde que se ejecutó la aplicación. Si la respuesta es afirmativa la aplicación se hace visible, pues es a partir de este instante cuando se puede ofrecer ayuda al alumno. Si la respuesta es negativa, se permanece a la espera y se vuelve a comprobar pasado un tiempo determinado.

La decisión de que la aplicación no se haga visible hasta detectar una compilación se debe a que se entiende que, hasta que esto no ocurra, los alumnos están realizando otras tareas (quizás aún programando) de las que no se puede ofrecer ayuda. Mientras que no se produzcan errores de compilación no se precisa de ayuda para resolverlos

El tiempo de espera hasta volver a comprobar si se ha producido una compilación, se simula mediante el uso de hilos. El método Thread.sleep suspende el subproceso actual durante el número de milisegundos especificado; en nuestro caso serán 2000 milisegundos o, lo que es lo mismo, 2 segundos. Es muy importante definir un tiempo de espera adecuado. Si es muy pequeño la aplicación consumirá muchos recursos pero, si es demasiado grande,

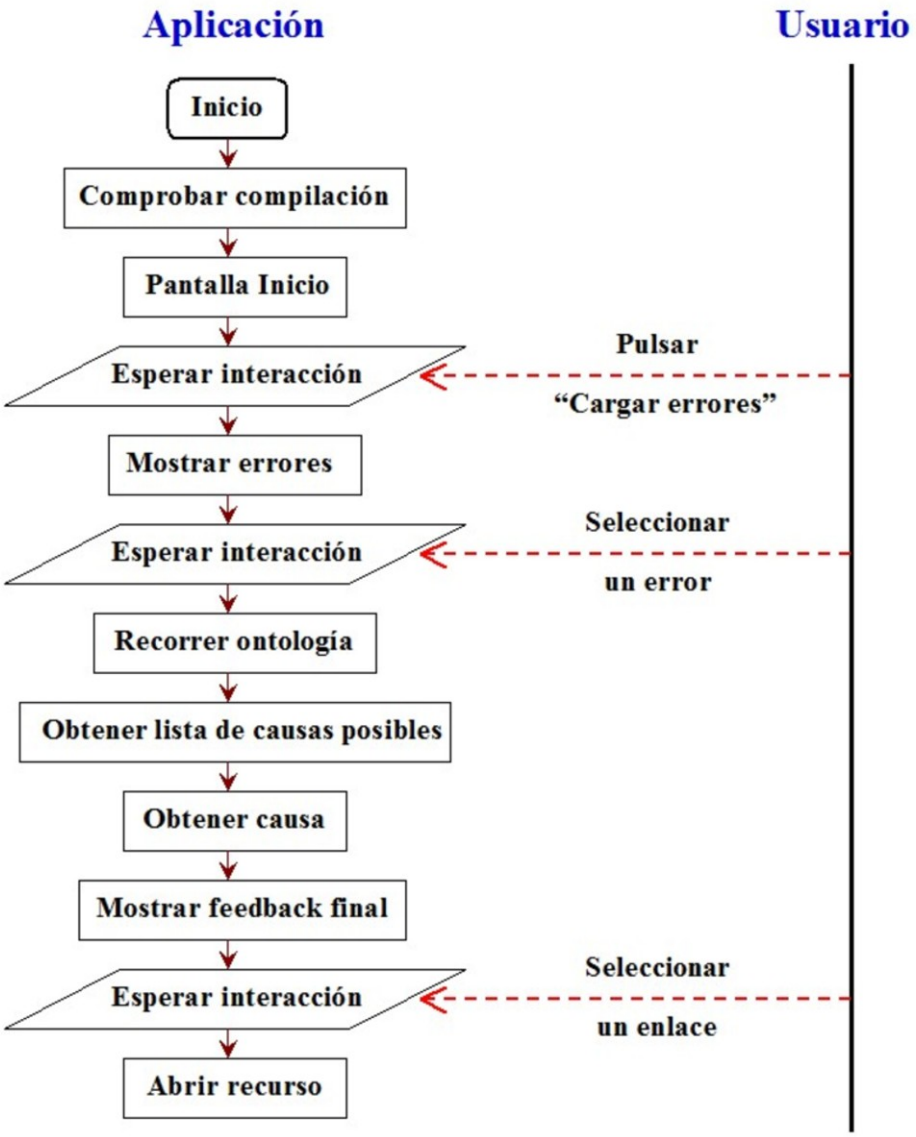


Figura 4.20: Diagrama de flujo del funcionamiento principal de la aplicación

el tiempo de reacción entre que se produce una compilación y se visualiza la aplicación sería muy elevado, y podría hacer esperar demasiado al alumno.

Para detectar si se ha producido una compilación, usamos la herramienta 'gcc' que incluye la Máquina Virtual. Dicha herramienta escribe información en un fichero de texto llamado 'gcc' cada vez que se produce una compilación.

Cada compilación producida desde línea de comandos se identifica por comenzar por una línea con la palabra clave '-BEGIN' y terminar con la sentencia '-END'. Entre ambas líneas se recoge la salida de la compilación; es decir, lo que se muestra en pantalla tras ejecutar la compilación de ficheros escritos en lenguaje de programación C. Un ejemplo se puede observar en la figura 4.21. En él se han producido numerosos errores. Si la compilación se hubiera realizado con éxito y no se hubiera producido ningún error o aviso, las líneas

intermedias no existirían y la siguiente línea a la que comienza por '-BEGIN' sería '-END'.

```
-BEGIN 1 37 0 2013-YY-XX HH:MM:SS /usr/bin/gcc fichero1.c
fichero2.c
In file included from fichero1.c:1:0:
fichero1.h:44:6: error: dos o más tipos de datos en los
especificadores de la declaración
fichero1.c: En la función 'nombreFunción'
fichero1.c:13:5: error: tipos en conflicto para 'nombreFunción'
fichero1.c: En la función 'nombreFunción':
fichero1.c:15:15: error: argumento de tipo inválido de '->' (se
tiene 'nombre')
fichero1.c: En la función 'nombreFunción':
fichero1.c:22:15: error: petición del miembro 'nombreCampo' en
algo que no es una estructura o unión
fichero1.c:43:7: error: redefinición de 'nombreFunción'
fichero1.c: En la función 'nombreFunción':
fichero1.c:71:3: error: sólo se permiten las declaraciones
iniciales del bucle 'for' en modo C99
fichero1.c: En el nivel principal:
fichero1.c:144:6: error: tipos en conflicto para 'nombreFunción'
fichero2.h:44:6: error: dos o más tipos de datos en los
especificadores de la declaración
fichero2.c: En la función 'nombreFunción':
fichero2.c:31:3: error: faltan argumentos para la función
'nombreFuncion'
fichero2.c: En la función 'nombreFunción':
fichero2.c:46:9: error: puntero deferenciado a tipo de dato
incompleto
-END
```

Figura 4.21: Fragmento del archivo 'gcc' que recoge los errores de compilación de los alumnos

Una vez conocemos el funcionamiento de la herramienta 'gcc', para saber si se ha producido una compilación es suficiente con comprobar si el número de líneas del fichero 'gcc.txt' ha aumentado, pues cada vez que esto ocurre se escriben dos o más nuevas líneas.

La ejecución de esta tarea inicial a la que hemos llamado 'Comprobar Compilación' se define en el diagrama de flujos de la figura 4.22. En ella podemos ver como cada 2 segundos se comprueba si ha aumentado el número de líneas del fichero 'gcc.txt' (lo que, como ya se ha explicado, significaría que se ha producido una nueva compilación). Esta acción se repite hasta que la respuesta es afirmativa, momento en el que creamos un objeto de la clase 'Aplicación' que hará visible la aplicación. Además se pone el valor 1 a la variable iniciado, que es una forma de indicar al software que la aplicación ya es visible.

Desde el momento en que la aplicación se hace visible, el alumno ya puede interaccionar. En primer lugar aparecerá una pantalla de inicio para avisar al usuario de que la aplicación está disponible y, tras pulsar el botón 'Cargar errores', se comienza con la tarea de 'Mostrar errores'. En este punto se muestra al alumno una lista con todos aquellos errores ocurridos en su última compilación y sobre los que se le puede obtener ayuda. El resto de errores no aparecen y, en el caso de que no se hayan producido errores o no se pueda ofrecer ayuda sobre ellos, se muestra el correspondiente mensaje informativo. En esta situación no se pueden consultar causas de error, pero el resto de funcionalidades de la aplicación si están disponibles, por lo que el usuario podría, por ejemplo, consultar teoría sobre ciertos conceptos que puedan interesarle (esta funcionalidad será explicada más adelante).

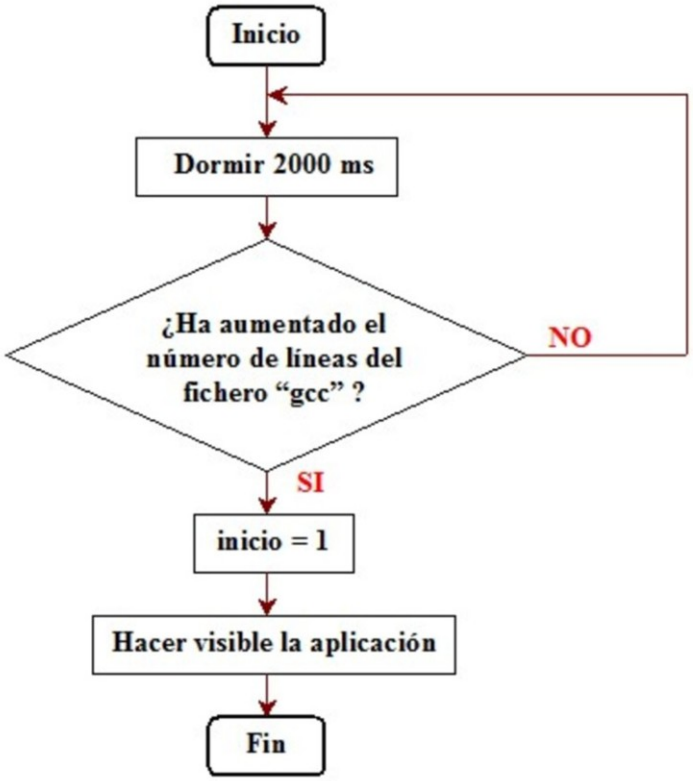


Figura 4.22: Diagrama de flujo de la tarea 'Comprobar Compilación'

Para poder identificar y mostrar los errores ocurridos en la última compilación, usamos la herramienta 'gcc' que incluye la Máquina Virtual y que es capaz de escribir en las últimas líneas de un fichero llamado 'gcc.txt', la descripción de los errores ocurridos en la última compilación. Por lo tanto, con tan sólo leer las líneas adecuadas de este fichero, la aplicación dispondrá de un listado de los errores.

Tras mostrar los errores, la aplicación permanece a la espera hasta que el usuario elija aquel sobre el que desea obtener conocimiento. Los errores se tratan de forma individual, por lo que el resto serán ignorados.

Cuando un error es seleccionado, se procede a cargar y 'Recorrer su ontología'. La finalidad de esta tarea es ir recorriendo la ontología asociada al error seleccionado de forma descendente, hasta llegar a las clases que se encuentran más abajo en la jerarquía de clases y que se corresponden con las diferentes causas específicas para ese error.

El razonador desarrollado para poder ir recorriendo la ontología e ir seleccionando el camino correcto, se basa en la interacción con el alumno. La idea es que cuando nos encontremos con ramificaciones se pregunte al alumno sobre que opción tomar. Entendemos como ramificaciones a las situaciones en las que, recorriendo la ontología hasta identificar causas finales, nos encontramos con clases que provocan varios caminos según diferentes valores de una propiedad. La lógica de esta tarea se puede consultar en la figura 4.23.

Para explicar de forma detallada este comportamiento, vamos a tomar como ejemplo el

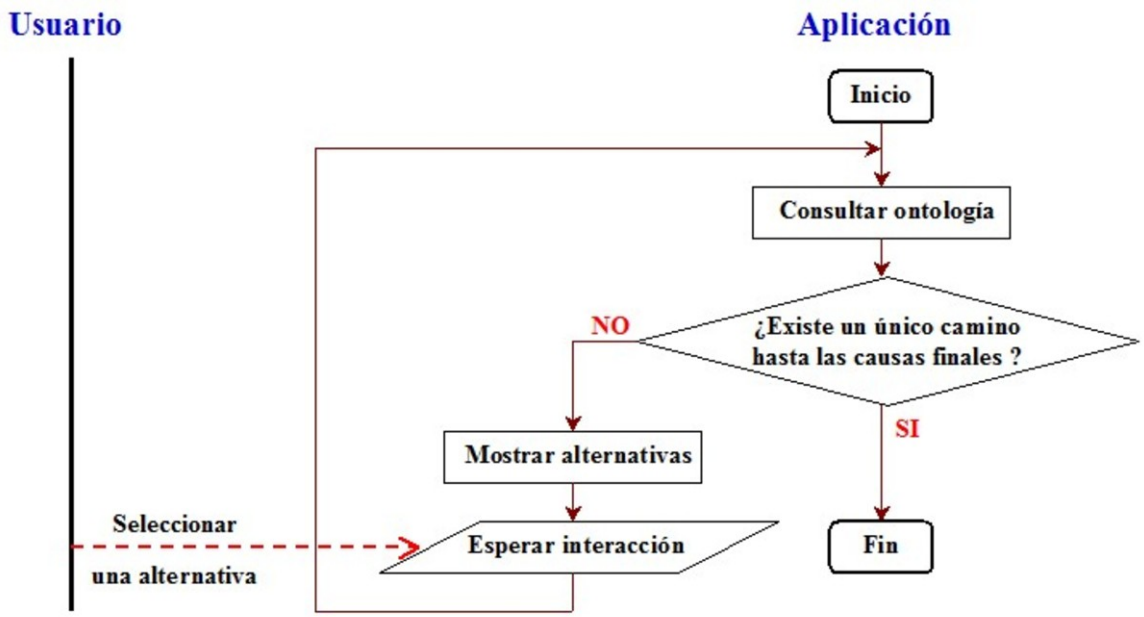


Figura 4.23: Diagrama de flujo de la tarea 'Recorrer ontología'

error 'Conflicting_types' cuya ontología se encuentra en la figura 4.14. En primer lugar nos encontramos con dos ramificaciones, ya que el error puede ocurrir tanto en una variable o como en una función. Como nuestra aplicación no tiene forma de conocer si el nombre del elemento sobre el que se ha producido el error es una causa u otra, se genera la pantalla de la figura 4.24 para que el alumno seleccione la opción correcta.

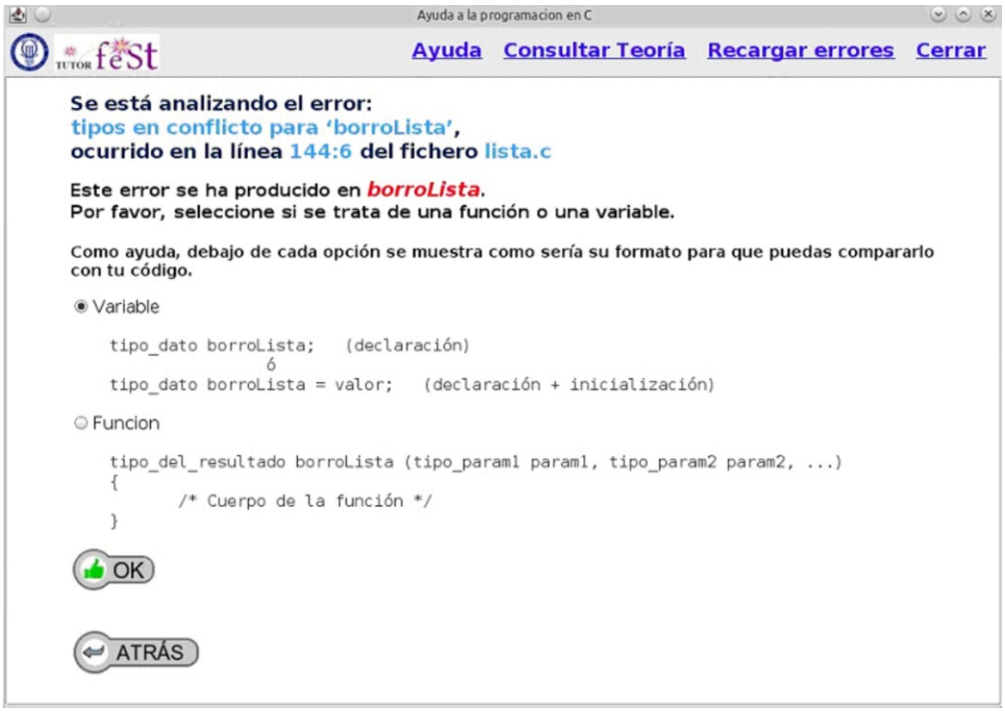


Figura 4.24: Ejemplo de interacción con el alumno para recorrer la ontología

Gracias a la elección del usuario, el razonador ya puede saber que camino tomar para ir discerniendo la causa que produjo dicho error de compilación. Estas preguntas se irán sucediendo hasta llegar al final de la ontología donde se encuentran las posibles causas.

Para identificar que consultas hemos de realizar en cada una de las ramificaciones, se ha definido un array que guarda, de forma personalizada para cada error, las propiedades que intervienen en cada uno de estos puntos. Además se cuenta con un contador que nos va a identificar en todo momento en que punto nos encontramos.

En nuestro ejemplo, dicho array sería el mostrado en la figura 4.25. La primera ramificación ocurre en la clase 'Conflicting_types_error' que se divide en dos posibilidades según el valor de la propiedad 'puede_ocurrir_en'. La segunda y última sucede en la clase 'Funcion' según la propiedad 'puede_ser_causado_por'.

```
array[0]="puede_ocurrir_en";
array[1]="puede_ser_causado_por";
```

Figura 4.25: Array de propiedades para el error 'Conflicting types'

Según la propiedad y el punto de la ontología en el que nos encontremos, se realizará una determinada consulta SPARQL para obtener las alternativas posibles. Las consultas están lo más generalizadas posible para que se puedan reutilizar en otros errores, por ello se usan variables como error, clase, o causa, que tomaran valores específicos para cada situación.

Volviendo a nuestro ejemplo, en la primera ramificación la consulta a realizar sería la definida en la figura 4.26. Donde la variable error tomaría el valor de 'Conflicting types' y la variable propiedad representa el valor definido en la posición del array que se corresponde con el punto en el que nos encontramos. En este caso array[0] que vale 'puede_ocurrir_en'.

```
     "PREFIX vcard: <http://www.w3.org/2002/07/owl#> "+
     "SELECT ?x " +
     "WHERE {?y vcard:onProperty
<http://www.semanticweb.org/ontologies/"+error+".owl#"+
propiedad+"> ."+
     "?y vcard:someValuesFrom ?x .}";
```

Figura 4.26: Ejemplo de consulta en una ramificación

Esto se sucedería para todas las ramificaciones existentes menos la última (último valor del array), donde existe una clase que está relacionada de forma directa (mediante una propiedad) con otras que son subclases de la clase 'Causas_finales', y que por tanto son las representan las posibles causas de error para una situación específica que genera el código del alumno. En este caso el tipo de consulta SPARQL varía un poco y sería como el que se muestra en la figura 4.27. Ahora queremos consultar las causas finales asociadas a una clase a la que hemos llegado recorriendo la ontología bajo ciertos valores.

Una vez disponemos de una lista con las posibles causas que han podido causar el error, se procede a la tarea identificada como 'Obtener Causa' cuya finalidad es ir mostrando una

```
        "PREFIX vcard: <http://www.w3.org/2002/07/owl#> "+
        "SELECT ?x " +
        "WHERE {?nodo vcard:someValuesFrom ?y ."+
        "<http://www.semanticweb.org/ontologies/"+error+".owl#
        "+clase+"> vcard:equivalentClass ?nodo ."+
        "?y vcard:equivalentClass ?nodoy ."+
        "?nodoy vcard:someValuesFrom ?x ."+
        "?x <http://www.w3.org/2000/01/rdf-schema#subClassOf>
<http://www.semanticweb.org/ontologies/"+error+".owl#Causas
_finales> .}";
```

Figura 4.27: Ejemplo de consulta sobre causa finales específicas

a una todas las posibilidades hasta que el alumno identifique su caso. Su diagrama de flujo se muestra en la figura 4.28.

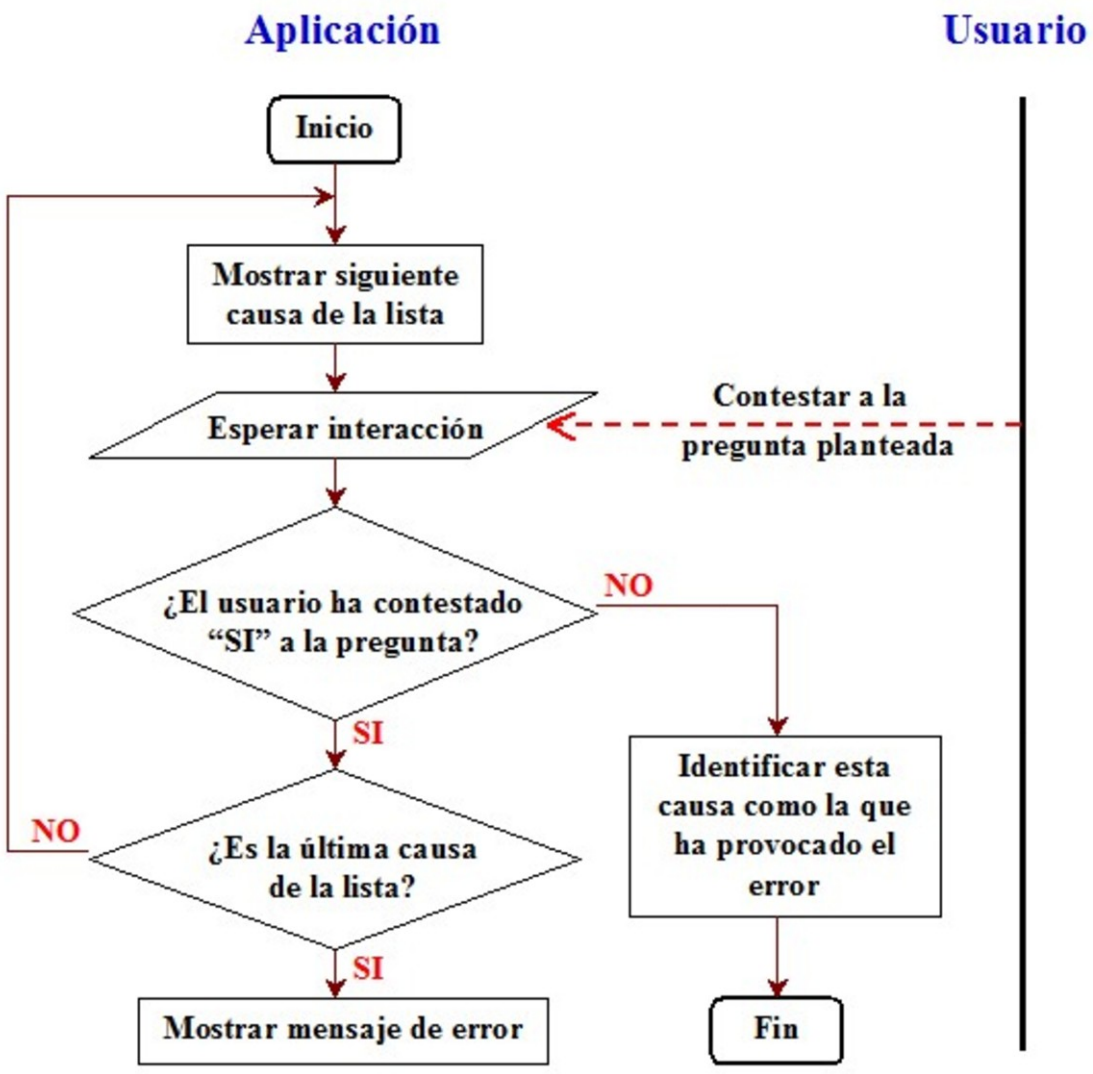


Figura 4.28: Diagrama de flujo de la tarea 'Obtener causa'

Un ejemplo de la pantalla que muestra al usuario una posible causa de error, se puede ver en la figura 4.39 del apartado 4.4 donde se muestra toda la interfaz gráfica de la aplicación. En ella aparece una pregunta dirigida al usuario a la que se debe contestar 'Si' o 'No'. Si la respuesta es afirmativa es que el código es correcto y esa no es la causa del error, por lo que se procede a mostrar la siguiente posible causa de la lista. En el supuesto en el que esa fuera la última causa de la lista, se mostrará el correspondiente error informando de que no se ha encontrado la causa del error (este mensaje se puede ver en la figura 4.42). El hecho de que la respuesta a alguna de estas preguntas sea negativa significa que algo no es correcto y por lo tanto esta es la causa que ha provocado el error en el código que ha compilado el alumno.

Como todo el texto que conforma la pregunta asociada a cada causa es muy extenso, pues en muchas ocasiones presenta explicaciones y/o ejemplos, se ha decidido almacenar esta información en un fichero externo para evitar saturar el SW. Además con esta medida podemos modificar el texto de forma rápida y sin necesidad de volver a compilar el código completo de la aplicación.

Dicho fichero externo mencionado se llama 'comentarios.txt', y almacena el texto o comentario de cada una de las posibles causa de cada uno de los posibles error. Para acceder a la parte del fichero concreta donde se encuentra el comentario de una causa específica, debemos hacer una consulta SPARQL para obtener una referencia que se encuentra en la propia ontología del error.

En las ontologías definidas, todas las clases que derivan de 'Causas_finales' contienen una propiedad llamada 'comment' que almacena una referencia con la siguiente estructura: 'Más información sobre esta causa en el fichero 'comentarios' entre las líneas inicio:XX fin:YY'; donde se indica la línea de inicio y de fin del fichero 'comentarios.txt' donde se encuentra el comentario completo asociado a esa causa.

Por lo tanto, para visualizar este tipo de pantalla y mostrar las preguntas sobre cada una de las causas posibles que nos ayudaran a identificar cual es la verdadera causa de error, también se ha hecho uso de conocimiento contenido en la propia ontología del error. Las consultas SPARQL para obtener la referencia necesaria para poder extraer la pregunta asociada a una causa deseada, se define en la figura 4.29. En ella se pregunta sobre la propiedad 'comment' de una determinada causa, que es donde se almacena la referencia.

```
"SELECT ?x " +
"WHERE
{<http://www.semanticweb.org/ontologies/"+error+".owl#"+c
ausa+"> <http://www.w3.org/2000/01/rdf-schema#comment> ?x
.}";
```

Figura 4.29: Consulta sobre la propiedad 'comment' que poseen ciertas clases de la ontología

Una vez identificada la verdadera causa de error, se inicia la tarea de 'Mostrar el feedback final' que muestra al usuario una pantalla, como las que aparecen en las figuras 4.40 y 4.41, con conocimiento que le puede ayudar a solucionar su error y facilitarle el aprendizaje. El conocimiento que ofrece este feedback se compone de:

- Una breve explicación sobre la causa que ha provocado el error de compilación y una propuesta de solución.

  Estos datos se extraen, al igual que la pregunta que se muestra al usuario para identificar la causa verdadera de error durante la tarea anterior de 'Obtener causa', del comentario almacenado en el fichero 'comentario.txt' obtenido con anterioridad. El formato de un comentario asociado a una causa específica, sigue la estructura mostrada en la figura 4.30. El texto incluye una serie de etiquetas que serán interpretadas por el software para proporcionar a cada parte el color y la fuente adecuados.

  La explicación sobre la causa que aparecerá en el feedback, se identifica como el texto que sigue a la palabra clave 'ERROR:' y la propuesta de solución se encuentre a continuación de la palabra clave 'SOLUCIÓN:'.

  Las líneas contenidas entre el inicio de la referencia y la línea que comienza por la palabra 'ERROR:', se corresponden con el texto completo que forma la pregunta asociada a esa causa y que se ha mostrado al usuario en la tarea anterior.

```
<titulo>¿El tipo de los argumentos declarados en el prototipo de
la función <nombre> coincide con<br>
los de la definición de dicha función?<NOtitulo><br>
Por ejemplo:<br>
<br>
<verde>Bien:<NOverde><br>
Prototipo:<br>
           void <nombre> (int a);<br>
Definición:<br>
           void <nombre> (int a)<br>
           {<br>
                /* Cuerpo de la función */<br>
           }<br>
<br>
<rojo>Mal:<NOrojo><br>
Prototipo:<br>
           void <nombre> (int a);<br>
Definición:<br>
           void <nombre> (char a)<br>
           {<br>
                /* Cuerpo de la función */<br>
           }<br>
ERROR: El error se ha producido porque el tipo de los argumentos
declarados en el protipo<br>
de la función <nombre> no coincide con los de su definición.<br>
SOLUCIÓN: Debe poner a cada uno de los argumentos de la
definición de la función <nombre>,<br>
el mismo tipo que se define en su prototipo.<br>
Puede cambiar el nombre de los argumentos, pero no el tipo o el
número.<br>
RELACIONADO CON: Definicion_funcion Prototipo_funcion
Parametros_funcion
```

Figura 4.30: Ejemplo de comentario sobre una causa específica tomada de un fragmento del fichero 'comentarios.txt'

- Listado de enlaces a aquellas páginas web (de entre todas las que conforman el temario de la asignatura 'Arquitectura de Sistemas') que ofrecen conocimiento sobre la causa específica de los errores.

Para obtener este listado se deben llevar a cabo diversas operaciones. En primer lugar se extrae, del comentario asociado a esa causa, un listado de conceptos con los que se relaciona esa causa. Estos se declaran a continuación de la palabra clave 'RELACIONADO CON:', y deben llamarse igual que las clases de la ontología sobre el lenguaje de programación C que los representa. Por lo tanto también han de coincidir con los nombres de clases usados en el etiquetado de recursos.

Una vez disponemos de los conceptos con los que se relaciona nuestra causa de error, se realiza una consulta SPARQL independiente para cada uno de ellos. De esta forma obtenemos una respuesta con los recursos que tratan sobre cada uno de estos conceptos. Dicha consulta tendría el formato que se muestra en la figura 4.31.

```
  "SELECT ?x " +
  "WHERE {?x <http://purl.org/dc/elements/1.1/label> \""+concepto+"\"
.}";
```

Figura 4.31: Consulta SPARQL para obtener recursos que tratan sobre un determinado concepto

El siguiente paso es transformar el conjunto de respuestas obtenido de todas las consultas sobre todos los conceptos, a una lista de enlaces que ofrecer al alumno. Como sucede en la mayoría de recomendadores actuales, se decidió que los enlaces que se muestran aparezcan ordenados por orden de prioridad, de forma que el primer enlace es el más relevante y el último el de menor importancia. O lo que es lo mismo en muestra aplicación, el primero es el que trata la causa del error de forma más detallada y concreta.

Otra decisión que se ha tomado sobre la visualización final de los enlaces, es que en primer lugar sólo aparezcan los 4 enlaces más relevantes para evitar la saturación del usuario. Pero existe la opción de 'Mostrar más enlaces' para que el usuario pueda ver y consultar todos los recursos obtenidos si así lo desea.

Las respuestas obtenidas de las consultas individuales sobre cada uno de los conceptos relacionado con la causa del error, se aúnan en un único listado. De forma que los recursos que sólo tratan sobre uno de estos conceptos, tendrán una única aparición. Pero si un recurso trata sobre dos de ellos, aparecerá en dos ocasiones ya que se ha devuelto como respuesta de dos consultas. Y así sucesivamente hasta poder aparecer un número de veces igual al total de conceptos.

Teniendo en cuenta este comportamiento, se va a proporcionar prioridad a los enlaces colocándolos según el número de apariciones en el listado, los que mas veces están repetidos se colocan en las primeras posiciones. Esta decisión se debe a que consideramos que un enlace que trate sobre todos los conceptos relacionados con la causa, es muy probable que hable sobre la propia causa. Sin embargo aquellos que sólo están relacionados con uno de los conceptos, tendrán algo que aportar, pero en menor medida.

Los criterios desarrollados para clasificar los enlaces son los siguientes:

- 'Enlaces aconsejados para consultar y comprender las causas del error': enlaces que aparecen repetidos, al menos una vez, en el listado. Se considera que son estos los que tratan de forma más específica la propia causa del error.

- 'Enlaces a conceptos y elementos que aparecen en el error': son los enlaces que aparecen una única vez en el listado porque sólo se han devuelto como respuesta a una de entre todas las consultas realizadas. Por tanto consideramos que representan enlaces a páginas que tratan sobre algo relacionado con el error, pero no tratan la causa en sí.

- 'Enlaces a conceptos generales': para poder obtener estos enlaces se han de realizar consultas SPARQL a la ontología general sobre el lenguaje de programación en C.

  Al igual que en el etiquetado de recursos, los conceptos relacionados con las causas deben ser lo más específicos posibles para obtener búsquedas efectivas. Pero puede ser que recursos que tratan sobre temas más generales también aporten conocimiento sobre la causa del error.

  Para buscar en la ontología con todo el conocimiento sobre lenguaje de programación C el nombre de clases más generales relacionadas con los conceptos específicos, se realiza la consulta de la figura 4.32. La respuesta engloba a todas las clases que contienen una condición necesaria y suficiente (equivalent) que incluye al concepto.

  En el ejemplo del error 'Clonflicting_Types', los conceptos con los que se relacionan una de sus posibles causas son: 'Definicion_funcion', 'Prototipo_funcion' y 'Parametros_funcion'. Estas clases forman parte de la definición 'equivalentClass' de la clase 'Funcion', ya que cualquier miembro que tenga prototipo, definición de función o parámetros es necesariamente una función. Por lo que al realizar la consulta de la figura 4.32 usando como valor de la variable 'concepto' cualquiera de estos 3 conceptos, la respuesta devolverá el nombre de 'Funcion'.

  En esta categoría se añadirán todos los recursos que estén etiquetados con las clases generales obtenidas en esta consulta y que no estén incluidos en categorías anteriores.

```
     "PREFIX vcard: <http://www.w3.org/2002/07/owl#> "+
     "SELECT ?x " +
     "WHERE {?y vcard:someValuesFrom
<http://www.semanticweb.org/ontologies/2012/0/OntologiaC.ow
l#"+concepto+"> ."+
     "?x vcard:equivalentClass ?y .}";
```

Figura 4.32: Consulta SPARQL para obtener recursos de conceptos generales

En resumen, estos enlaces se corresponderán con enlaces a páginas que hablan sobre conceptos mucho más general pero relacionados con los responsables o implicados en el error, que también pueden aportar conocimiento interesante para el alumno.

No para todas las causas aparecen todas las categorías, pues hay situaciones en las que la causa sólo se relaciona con un concepto y por lo tanto el control de repeticiones no tiene sentidos. En otras ocasiones no habrá enlaces a conceptos generales porque todos los recursos devueltos en esta consulta ya se incluye en categorías anteriores.

Los enlaces finales que se muestran al usuario, deben aparecer clasificados en 'Teoría', con aquellos recursos que sólo ofrecen información teórica, y 'Ejercicios' con enlaces a recursos que ofrecen prácticas o ejercicios y no mero contenido teórico.

El criterio desarrollado para poder clasificar en 'Teoría' o 'Ejercicios' se basa en que los enlaces que contienen ejercicios deben estar etiquetados como tales, y tener una propiedad 'label' cuyo valor es 'Ejercicio'. Como vimos en el apartado 4.2, al etiquetar estos recursos se debe incluir la sentencia: <dc:label>'Ejercicio'</dc:label>. Gracias a ello, si realizamos una simple consulta SPARQL (como la que aparece en la figura 4.33) que devuelve todos los objetos que se relacionan mediante la propiedad 'label' con ese determinado recurso y se comprueba que la respuesta contiene el valor 'Ejercicio', podemos identificar ese recurso entre los que contienen ejercicios. Si la respuesta no devuelve el valor esperado, se trata se un enlace con contenido teórico.

```
        "SELECT ?x " +
        "WHERE {<"+enlace+">
<http://purl.org/dc/elements/1.1/label> ?x .}";
```

Figura 4.33: Consulta SPARQL para saber el si el contenido de un recurso es teórico o de ejercicios

Como tarea final para completar el funcionamiento principal de la aplicación, se debe ejecutar la tarea de 'Abrir recurso'. Los enlaces mostrados como feedback final son reales, de forma que si el usuario desea consultar alguna de ellos y pincha sobre él, se abre el navegador correspondiente y se carga el recurso.

Una vez explicado el flujo normal de funcionamiento de la aplicación, se debe mencionar que durante todo el proceso está disponible un menú superior principal (como el que aparece en la figura 4.46) que puede alterar este flujo y redirigir al usuario a otras tareas que serán especificadas a continuación:

- Obtener ayuda: redirige a una pantalla informativa que explica la interfaz y funcionalidades de la propia aplicación, así como la forma de interactuar con ella. Esta operación no altera el flujo de funcionamiento principal, por lo que al volver atrás la aplicación mostrará la pantalla anterior a la selección de este opción.

- Recargar Errores: retoma el punto de la aplicación donde se muestran los errores, actualizados a la última compilación. En la figura 4.20 este punto está representado por la tarea 'Mostrar Errores'

- Cerrar: es una forma de terminar de forma limpia con la aplicación. En realidad no existe ninguna diferencia con salir pulsando sobre el botón cerrar que incluye la propia ventana.

- Consultar teoría: herramienta complementaria, no planeada en los objetivos iniciales del proyecto pero que durante el diseño de la aplicación surgió como una forma sencilla de aportar una nueva útil funcionalidad. Con ella los alumnos pueden consultar los recursos web que tratan sobre uno o varios conceptos de entre los que aparecen en una lista (ver figura 4.43), y así ahorrarse una búsqueda manual. Su gran utilidad es que esta herramienta es capaz de mostrar los enlaces que tratan sobre varios de los conceptos seleccionados, ofreciendo así recursos que muy probablemente contienen conocimiento sobre la relación entre ellos.

La lógica para devolver los enlaces es muy parecida a la de los enlaces que aparecen como feedback a la causa del error. La pantalla que genera esta herramienta, se puede consultar en la figura 4.44. Como se puede ver, los enlaces resultantes aparecen divididos en Teoría y Ejercicios, y los primeros a subes en las siguientes categorías:

- 'Enlaces a páginas que relacionan varios de los conceptos seleccionados': como se realiza una consulta individual para cada uno de los conceptos seleccionados y posteriormente se aúnan todas las respuestas en un único listado, en esta categoría incluiremos los enlaces repetidos. Se colocarán de mayor a menor número de apariciones en el listado para dar prioridad a los recursos que tratan sobre más conceptos de entre todos los seleccionados.
- 'Enlaces a páginas que tratan sobre un sólo concepto de entre los seleccionados: enlaces que no aparecen repetidos porque están etiquetados como que sólo tratan sobre uno de los conceptos seleccionados. Una posible mejora sería indicar en cada enlace sobre que concepto en concreto, de entre todos los seleccionados, trata ese recurso. En este proyecto no se ha considerado que esto suponga un gran problema, ya que esta herramienta se trata de una funcionalidad secundaria y que siempre existe la posibilidad de realizar una consulta en la que sólo se seleccione un concepto concreto. Esta herramienta está más orientada a buscar relaciones entre conceptos a través de recursos que ofrezcan conocimiento sobre varios de ellos.
- 'Enlaces a conceptos generales': la lógica para obtener los enlaces de esta categoría es la misma que en el caso del feedback sobre la causa del error.

4.3.1. Estructuración del código

El proyecto completo se compone de 5 clases, cada una de ella orientadas a una funcionalidad específica. A continuación se procederá a explicar de forma detallada cada una de estas clases, así como la interacción entre ellas. Para completar esta información, en la figura 4.34 se representa un diagrama de clases de la aplicación.

- **LeerFichero.c**: se encarga de la lectura de ficheros externos para la obtención de información necesaria para la aplicación. Concretamente extrae información de dos ficheros explicados anteriormente en esta memoria, 'gcc.txt' y 'comentarios.txt'. El primer fichero se usa para obtener información sobre las compilaciones que realizan los alumnos. Para ello se han definido, por ejemplo, los siguientes métodos:
  - dameLineaUltimaComilacion(): devuelve el número que ocupa la última fila del fichero.
  - dameNumLineas(): devuelve el número de líneas totales del fichero.
  - getErroresCompilacion(): devuelve las líneas entre el última HEAD y el último END donde se encuentran los errores de la última compilación.

  El segundo fichero se usa para obtener información sobre una causa específica. El método leerComentario(), usando para ello las líneas de inicio y fin pasadas por parámetro, extrae del fichero las líneas que se refieren al comentario de la causa deseada.

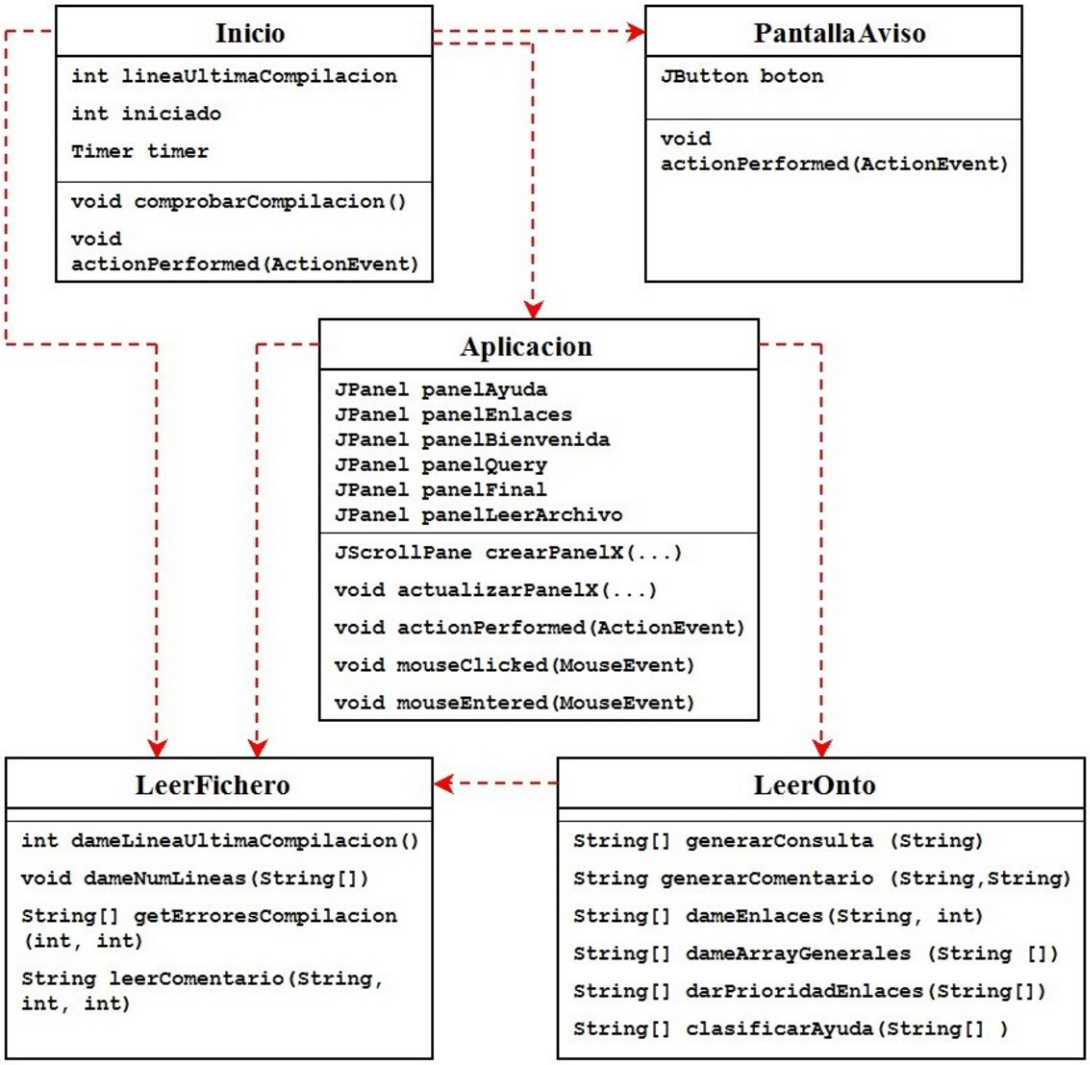


Figura 4.34: Diagrama de clases de la aplicación

- **Inicio.c**: es la clase que contiene el main del proyecto. Realiza dos funciones principales, por un lado inicia la tarea de 'Comprobar Compilación' cuyo diagrama de flujo se encuentra en la figura 4.22.

  El main inicialmente llama al método dameLineaUltimaComilacion() de la clase LeerFichero y guarda el valor devuelto por esta función en una variable llamada lineaUltimaCompilacion. Para comprobar si el número de líneas ha aumentado, realizamos una nueva llamada a este método y comparamos el valor devuelto con el almacenado en la variable lineaUltimaCompilacion. Si ha aumentado, se ha producido una nueva compilación y actualizamos dicha variable al nuevo valor.

  Cuando se produce una primera compilación, la variable iniciado que por defecto vale 0, cambiará su valor a 1.

  Para hacer visible la aplicación, tan sólo habrá que crear un objeto de la clase Aplicación.c que crea y actualiza toda la interfaz gráfica.

Por otro lado, esta clase se encarga de comprobar constantemente durante todo el tiempo que la aplicación permanezca en ejecución, si se ha producido una nueva compilación para lanzar el correspondiente aviso informativo. Para ello se usa un timer que genera un evento cada 3500 ms, el diagrama de flujo de las tareas que se realizan en cada uno de estos eventos del timer se muestran en la figura 4.35.

La lógica de estos eventos es la siguiente: en primer lugar se comprueba mediante el valor de la variable iniciado si se ha producido una primera compilación. El valor se esta variable será 0 si aún no se ha producido y por lo tanto no tiene sentido lanzar un aviso de nueva compilación, y 1 si ya se ha detectado una primera compilación y debemos controlar cuando se produce otra.

Si la variable iniciado vale 1, controlamos si se produce una nueva compilación usando la comparación de número de líneas anteriores y actuales explicada en la tarea 'Comprobar Compilación'. Si el número de líneas ha aumentado, se crea un objeto de la clase PantallaAviso cuya finalidad es precisamente generar una pantalla de aviso informando de que existe una nueva compilación.

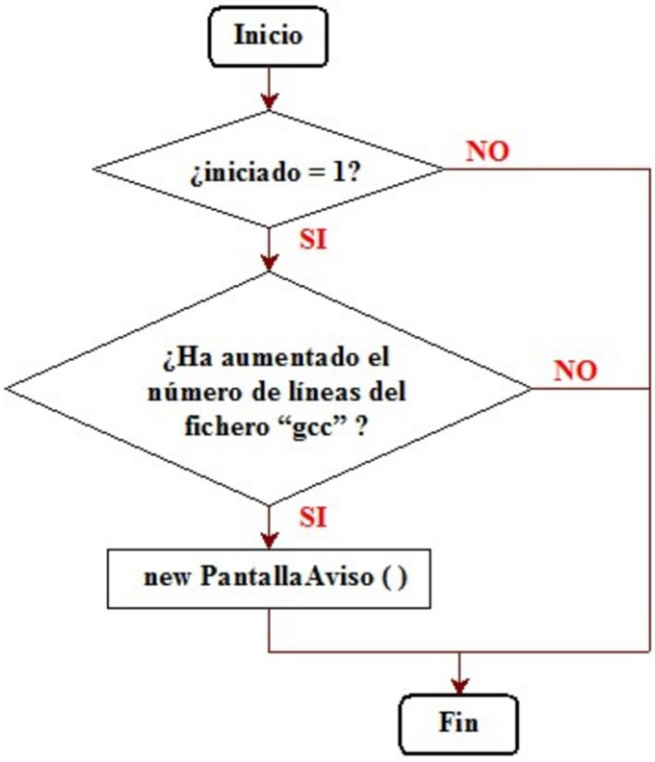


Figura 4.35: Diagrama de flujo de cada uno de los eventos del timer

- **PantallaAviso.c**: Implementa una ventana secundaria, por lo que está definida como una clase que extiende de JDialog. Dicha ventana no es más que un aviso que informa de que se ha detectado una nueva compilación y se debe pulsar el botón 'Recargar Errores' del menú principal para introducir en la aplicación los nuevos errores que se hayan podido producir en la última compilación.

Esta ventana secundaria tiene una funcionalidad meramente informativa, por lo que se cerrará tras pulsar su botón 'Aceptar' sin realizar ninguna otra acción. Aunque esta ventana no se cierre, cuando la aplicación detecte una nueva compilación, se creará un nuevo objeto de esta clase y aparecerá una nueva ventana de aviso para informar de la nueva compilación.

Para ver la estructura de esta ventana, se puede consultar la figura 4.45.

- **LeerOnto.c**: es la clase que se encarga de conectarse con la base de datos SDB y realizar todas las consultas SPARQL necesarias para la obtención de información. Además se encarga de obtener, clasificar y ordenar por prioridad los enlaces que se mostrarán en el feedback final. Sus métodos principales son:
  - generarConsulta: es capaz de conectarse a la base de datos, realizar la consulta pasada como parámetro y devolver un array con todas las respuestas obtenidas.
  - generarComentario: en primer lugar genera una consulta, como la especificada en la figura 4.29, para obtener las líneas de inicio y fin entre las que se encuentra el comentario asociado a una causa. Una vez obtenidas dichas líneas, se realiza una llamada al método leerComentario definido en la clase LeerFichero para obtener el comentario. Tras transformar una serie de etiquetas del texto en cambios de color o fuente, se devuelve la pregunta que ayudará al usuario a identificar la causa de su error.
  - dameEnlaces: este método realiza una consulta individualizada, como la especificada en la figura 4.31, para cada uno de los conceptos pasados por parámetro con los que se relaciona una determinada causa del error. Por último realiza un listado de enlaces con todos las respuestas obtenidas.
  - dameArrayGenerales: gracias a la consulta especificada en la figura 4.32, obtiene los enlaces a conceptos generales.
  - darPrioridadEnlaces: se encarga de ordenar un listado de enlaces recibido por parámetro basándose en el número de apariciones, los que mas veces están repetidos se colocan en las primeras posiciones.
  - clasificarAyuda: recibe un lista de recursos y, mediante una consulta como la definida en la figura 4.33, clasifica cada uno de ellos en enlaces a teoría o enlaces a ejercicios.
- **Aplicación.c**: es el único JFrame de la aplicación y desarrolla y actualiza toda la interfaz gráfica. Toda la aplicación se desarrolla sobre un panel de tipo CardLayout donde los distintos paneles se van mostrando unos encima de otros. La aplicación está desarrollada para que cada uno de estos paneles se corresponda con un estado concreto de la aplicación en la que se muestra cierta información y se permanece a la espera hasta que se produzca una nueva interacción con el alumno. Cada panel tiene sus propios elementos que garantizan la funcionalidad necesaria en cada punto. Existen los siguientes paneles: panelBienvenida (figura 4.36), panelLeerArchivo (figura 4.37), panelAyuda (panel que muestra información sobre el funcionamiento de la aplicación), panelEnlaces (figura 4.43), panelQuery (figura 4.39) y panelFinal (figura 4.40).

  Además, en la parte superior de la aplicación existe otro panel siempre visible que realiza la función de Menú principal.

Esta clase también incluye el manejador de eventos de la interfaz gráfica, por lo que desarrolla todo el control sobre la interacción con el usuario y es capaz de identificar el tipo de información que se debe visualizar en cada momento según el punto en el que se encuentre la aplicación. Para poder obtener todos los datos necesarios para ello, esta clase realiza numerosas llamadas a los métodos de las clases LeerFichero y LeerOnto, capaces de proporcionar toda la información requerida.

4.4. Interfaz

Figura 4.36: Pantalla de inicio

Para desarrollar una interfaz adecuada, se han tenido en cuenta los siguientes puntos:

- Interfaz lo más sencilla e intuitiva posible.
- Mensajes informativos para ir mostrando en todo momento las elecciones ya hechas para que el usuario no las tenga que recordar y sepa en todo momento en que punto se encuentra.
- Mensajes explicativos sobre el funcionamiento y las posibles opciones para que el usuario sepa como interactuar con el sistema.
- Se han intentado lograr las mejores respuestas de tiempo.

- En todo momento se muestra un botón 'Atrás' para que la reversibilidad de las acciones sea posible.
- Mensajes sobre el propio conocimiento para realizar preguntas lo menos técnicas posibles, pues se debe tener en cuenta que no todos los usuarios tienen el mismo nivel de conocimiento sobre el tema y que precisamente la finalidad de esta aplicación es ayudarles en el aprendizaje. Es importante adaptar el contenido a las características y necesidades al usuario, por lo que hay que guiarle en el proceso de consulta porque es posible que no tenga mucho conocimiento sobre el tema.
- El tamaño de la aplicación es de 920x650, lo suficientemente grande para poder mostrar de forma rápida y cómoda toda la información necesaria, pero sin ocupar toda la resolución de la pantalla de la máquina virtual para que el alumno no se sienta invadido y tenga espacio para navegar y consultar su código a la vez que interacciona con la aplicación.

A continuación se explicará de forma detallada como es la interfaz y cómo se desarrolla la interacción con el usuario. Se añaden diversas figuras con capturas de pantallas de la propia aplicación para visualizar su desarrollo.

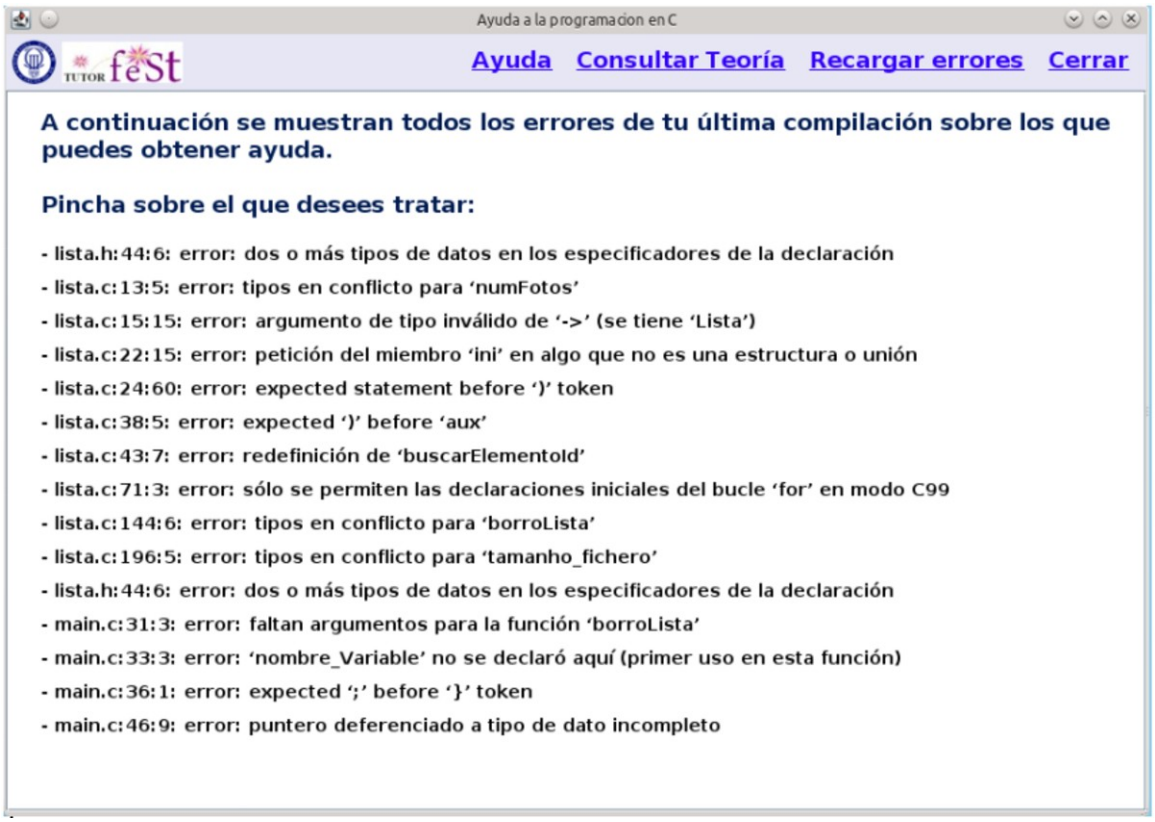


Figura 4.37: Ejemplo de pantalla que muestra todos los errores de compilación ocurridos en la última compilación y sobre los que se puede ofrecer ayuda

Cuando el usuario realiza una compilación y la aplicación detecta que se han producido errores sobre los que se puede ofrecer ayuda, la aplicación se hace visible. En primer lugar

aparecerá una pantalla de inicio, como la que aparece en la figura 4.36, para avisar al usuario de que la aplicación está disponible y, tras pulsar 'Cargar errores', se muestra la pantalla de la figura 4.37 con todos aquellos errores ocurridos en la última compilación sobre los que puedes obtener ayuda (el resto de errores, no aparecen).

Cuando el usuario selecciona el error que desea tratar, se inicia una interacción con el basada en una serie de preguntas que deberán ser comprobadas en el código que ha producido el error. Estas preguntas pueden ser de dos tipos:

1. Seleccionar una opción entre varias posibles. Un ejemplo de esta situación puede verse en la figura 4.38.

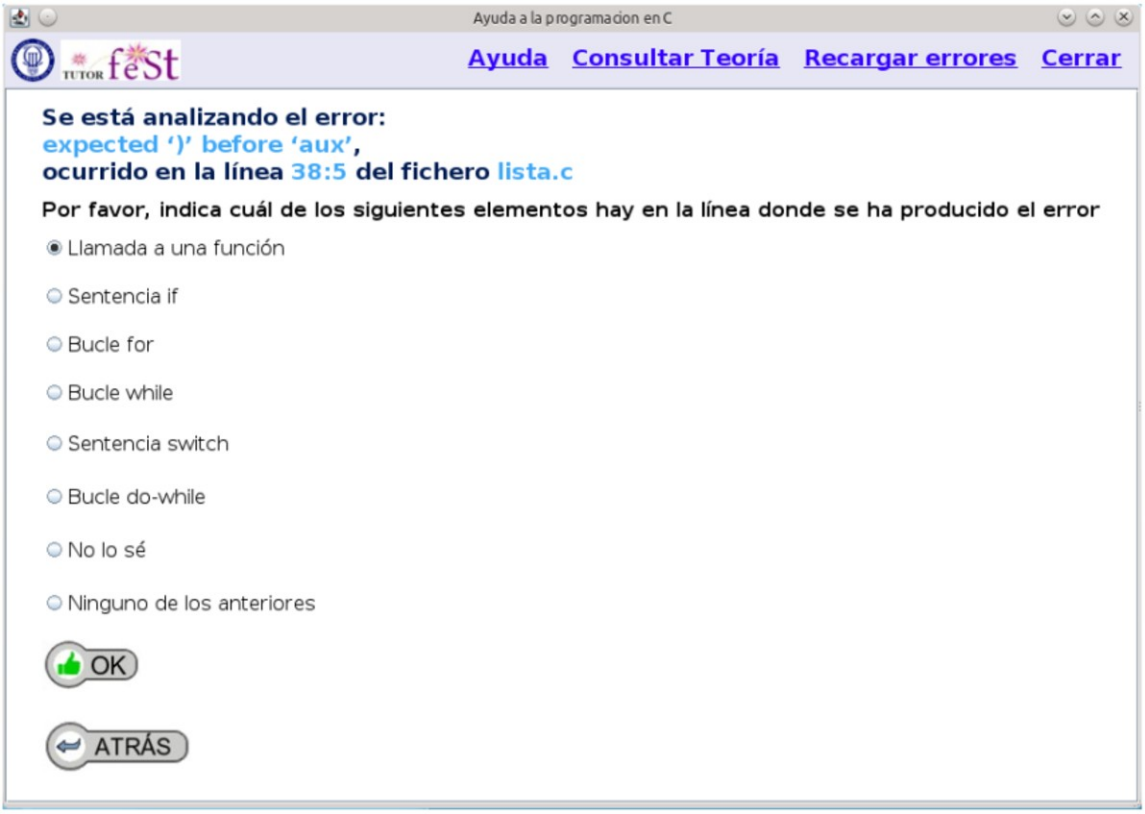


Figura 4.38: Ejemplo de pantalla que muestra interacción con el usuario mediante múltiples respuestas a una pregunta

2. Contestar 'SI' o 'NO' a la pregunta formulada en azul. Un ejemplo de esta situación puede verse en la figura 4.39.

   En ocasiones, a la hora de formular la pregunta se hace un supuesto (por ejemplo: en el supuesto de que en la línea sobre la que se ha producido el error, haya un bucle for). Si este supuesto no se corresponde con el caso particular del código del alumno (por ejemplo: en la línea sobre la que se ha producido el error, no hay un bucle for), este deberá pulsar 'SI' para pasar a la siguiente pregunta sobre una nueva posible causa de error.

Cuando la respuesta a alguna de estas preguntas sea 'NO', se habrá encontrado la causa del error y se visualizará otro tipo de pantalla con la CAUSA y SOLUCIÓN del mismo. Además se ofrecen enlaces a Teoría y Ejercicios relacionados con el error que ayudaran al alumno en su aprendizaje. Esta visualización de pantalla se muestra en la figura 4.40.

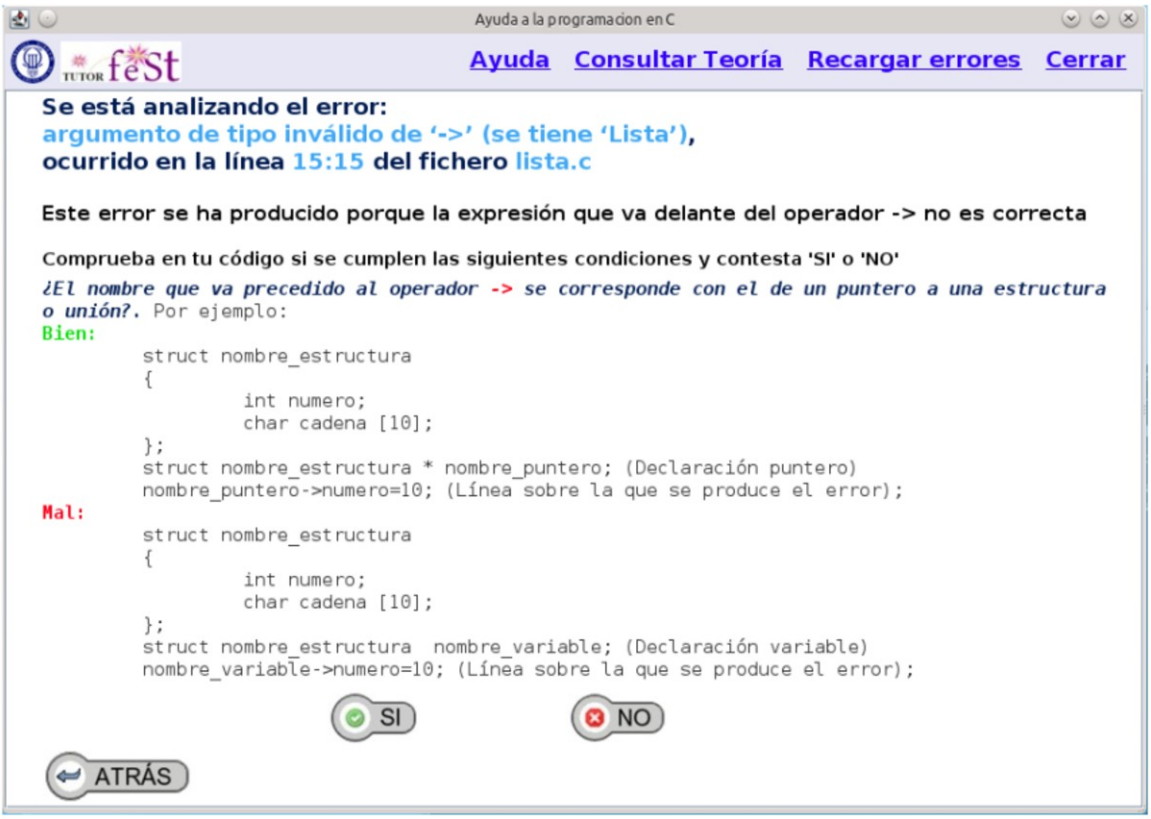


Figura 4.39: Ejemplo de pantalla que muestra interacción con el usuario mediante una pregunta a la que se debe contestar 'SI' o 'NO'

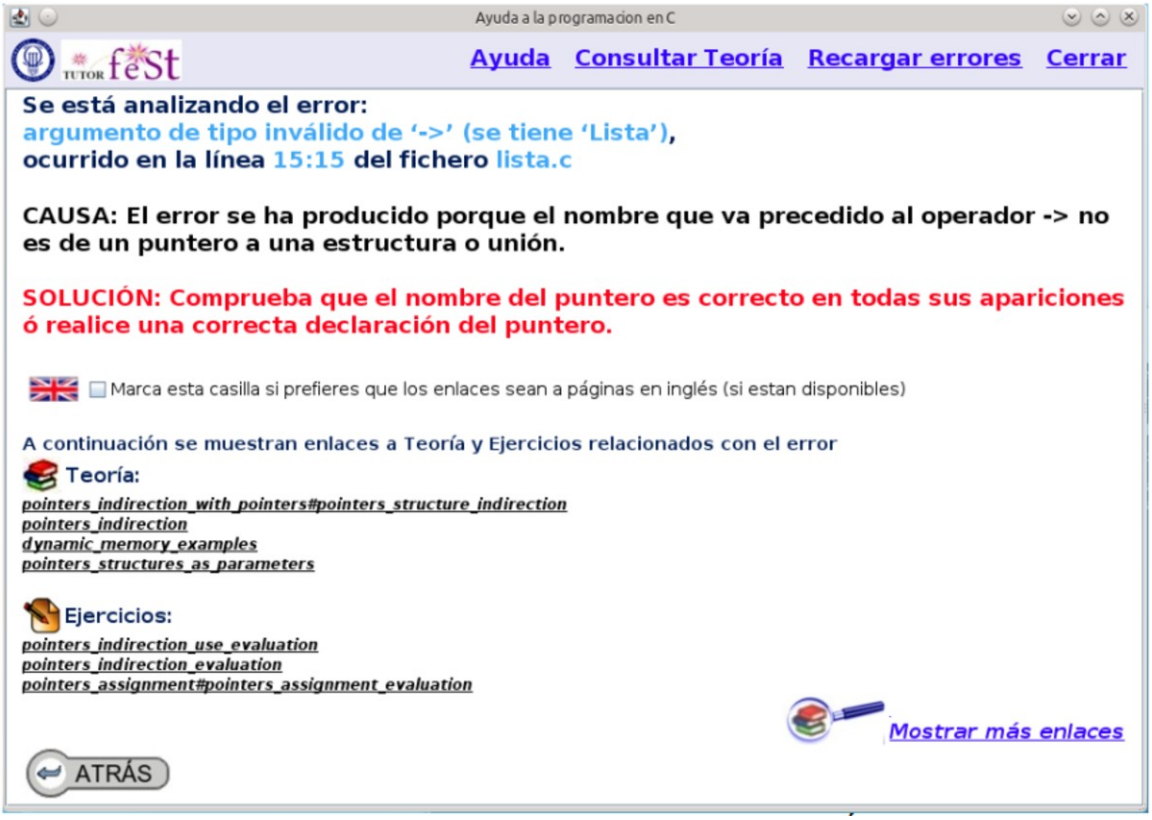


Figura 4.40: Ejemplo de pantalla con la CAUSA y SOLUCIÓN del error. Además se ofrecen enlaces a Teoría y Ejercicios

Si el alumno decide obtener más enlaces pulsando sobre el botón 'Mostrar más enlaces' que aparece en la captura de la figura 4.40, la pantalla se actualizará a la de la figura 4.41. Esta es una decisión de diseño para que el usuario no se sature con la presentación de muchos enlaces desde el principio y pueda caer en el error de no consultar ninguno.

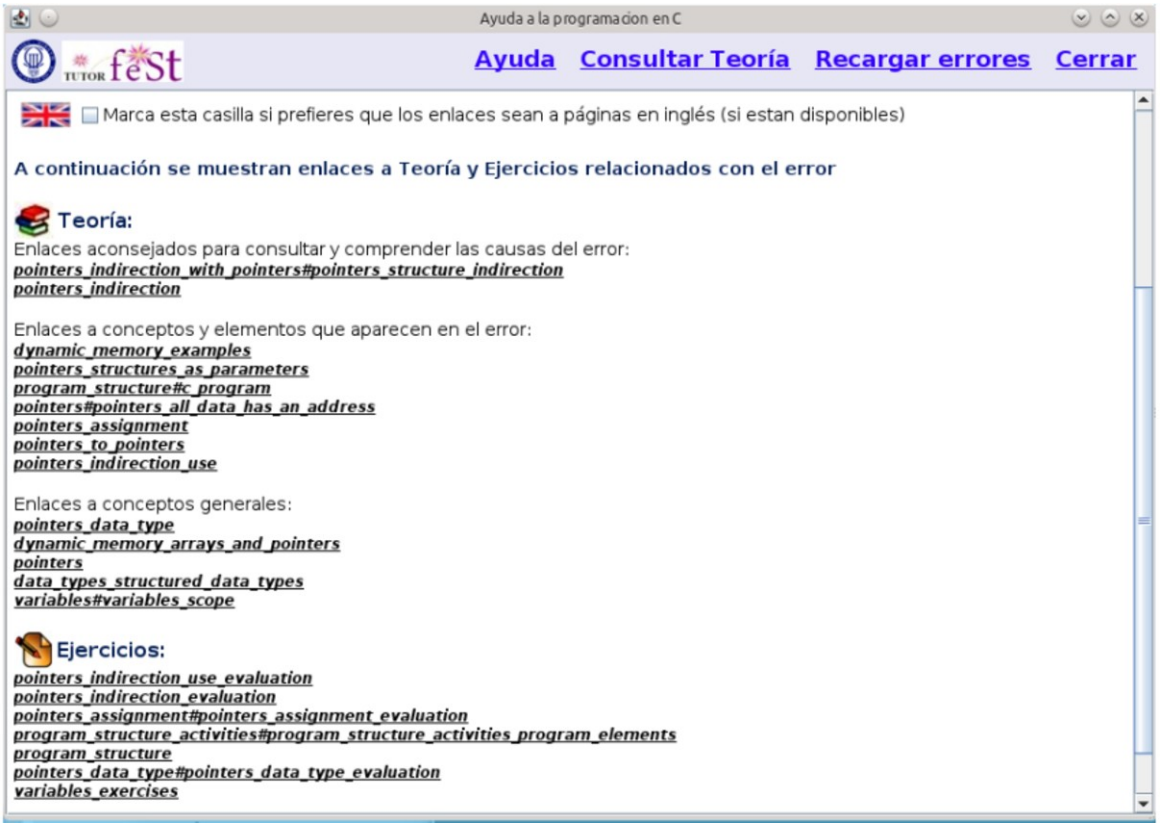


Figura 4.41: Ejemplo de pantalla con todos los enlaces a Teoría y Ejercicios relacionados con el error de compilación

Ahora, los enlaces a Teoría pueden aparecer divididos en las siguientes categorías, no en todos los casos aparecen todas las categorías:

- 'Enlaces aconsejados para consultar y comprender las causas del error': enlaces a páginas que hablan de forma concreta sobre la causa que ha provocado el error.
- 'Enlaces a conceptos y elementos que aparecen en el error': enlaces a páginas que hablan sobre alguno de los conceptos responsables o implicados en el error de compilación.
- 'Enlaces a conceptos generales': enlaces a páginas que hablan de forma mucho más general sobre los conceptos responsables o implicados en tu error.

En las figuras 4.40 y 4.41 también aparece la opción de señalar una casilla si el alumno desea que los enlaces le redirijan a páginas que contienen la información en inglés, siempre y cuando esta opción este disponible (por defecto, si no se señala dicha casilla, las páginas se mostraran en español).

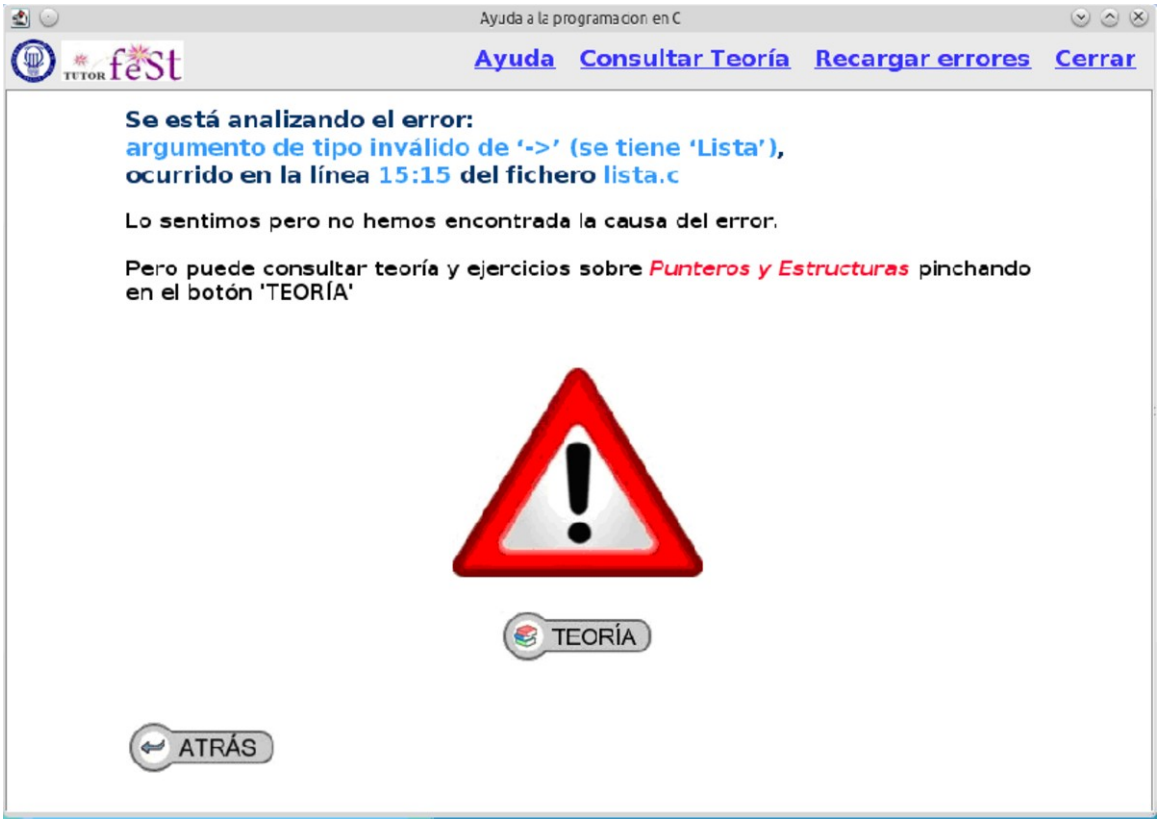


Figura 4.42: Ejemplo de pantalla cuando no se encuentra la causa del error

En ocasiones puede ser que no se encuentre la causa del error, en cuyo caso se mostrará una pantalla como la de la figura 4.42 con el mensaje de error correspondiente. Desde ella (pinchando en el botón 'TEORÍA) o pinchando en cualquier momento sobre 'Consultar Teoría' del menú superior, se accede a una herramienta desde la que se puede consultar Teoría y Ejercicios sobre uno o varios de los conceptos que aparecen en una lista. Esta lista se puede consultar en la figura 4.43.

Tras seleccionar los conceptos deseados y pulsar el botón 'OK' de la herramienta Consultar Teoría, se mostrará al usuario la pantalla de la figura 4.44 donde aparecen una serie de enlaces que pueden estar divididos en las siguientes categorías:

- 'Enlaces a páginas que relacionan varios de los conceptos seleccionados': enlaces a páginas que tratan sobre al menos dos de todos los conceptos seleccionados.
- 'Enlaces a páginas que tratan sobre un sólo concepto de entre los seleccionados
- 'Enlaces a conceptos generales': enlaces a páginas que hablan de forma mucho más general sobre uno o varios de los conceptos seleccionados.

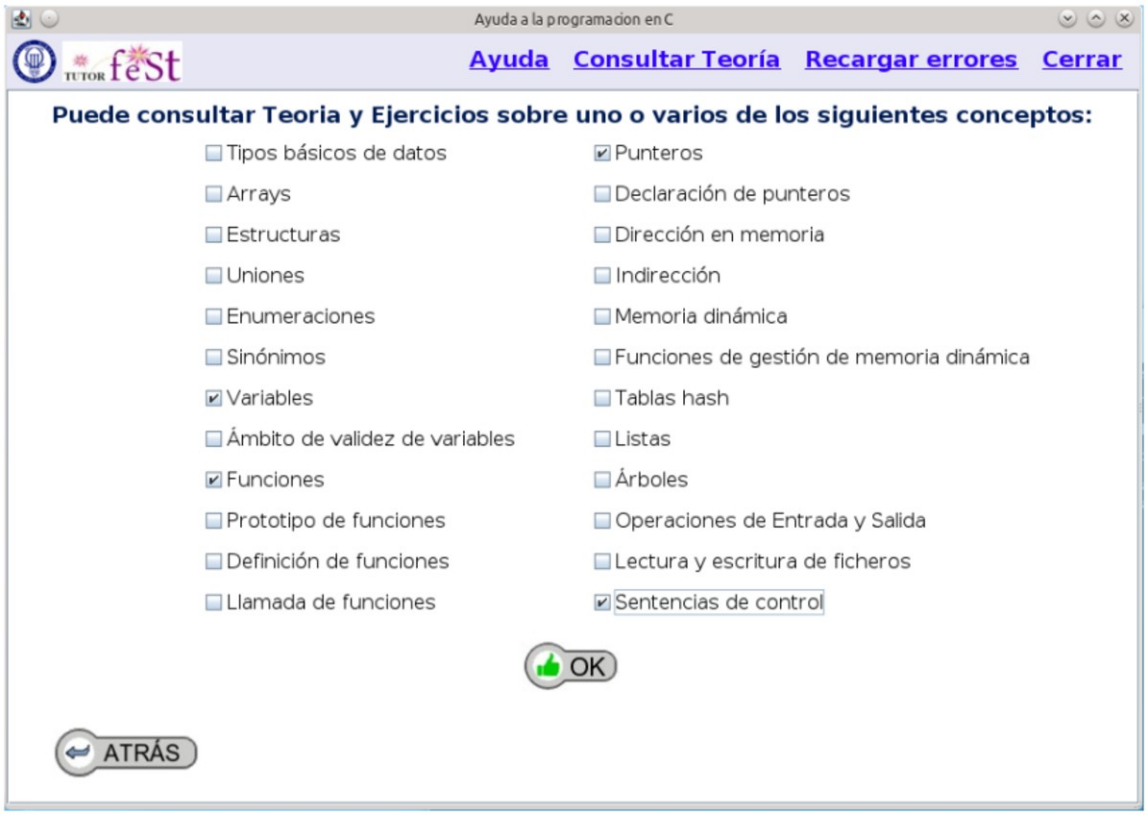


Figura 4.43: Ejemplo de pantalla que permite consultar enlaces sobre los conceptos listados

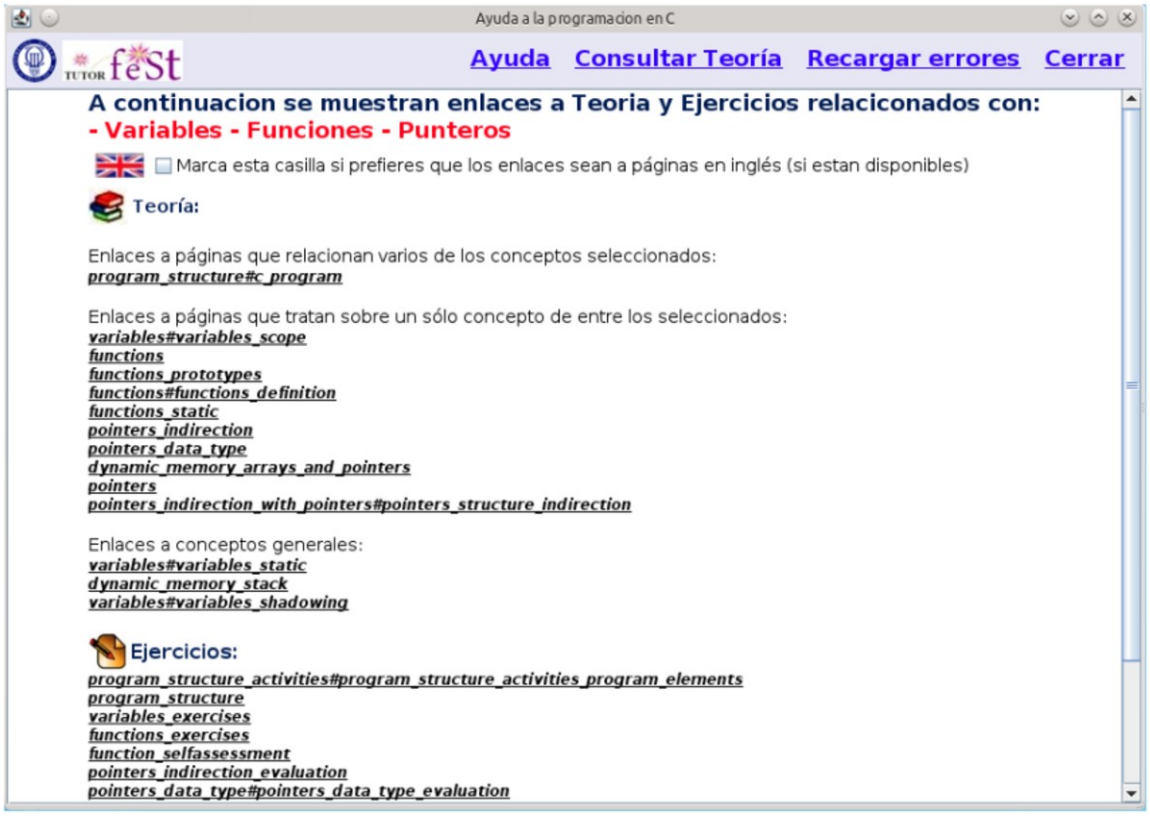


Figura 4.44: Ejemplo de pantalla con enlaces sobre los conceptos seleccionados

Si mientras te encuentras en cualquier punto de esta aplicación, se detecta una nueva compilación, se mostrará el aviso de la figura 4.45.

Figura 4.45: Pantalla de aviso de nueva compilación

Para cargar los errores de tu última compilación, ya sea porque se ha producido una nueva o porque deseas volver a comenzar con la búsqueda de la causa de un error, debes pinchar sobre 'Recargar errores' del menú principal (ver figura 4.46). Este menú superior está siempre visible en pantalla para que sus opciones estén disponibles en todo momento. En él también aparece la opción de ayuda, que muestra al usuario información sobre el funcionamiento de la aplicación.

Respecto a la estética de la aplicación, se ha optado por una fuente y colores similares a

los utilizados en la documentación de la asignatura 'Arquitectura de Sistemas' para que los alumnos asocien su vinculación a ella.

Figura 4.46: Menú superior

Todos los logos e imágenes que aparecen en la interfaz gráfica (así como en el contenido de esta memoria) son de creación propia, para evitar problemas de copyright. Además, se decidió crear un nombre y logo distintivos para la aplicación. La idea que surgió fue de dar un nombre divertido (para atraer a los alumnos) cuyas letras se pudieran usar como siglas de una frase que describiera los objetivos de la aplicación. Finalmente se decidió llamar a la aplicación 'TUTOR fest', donde la palabra 'fest' a su vez significa 'Feedback tool to compilation Errors based on Semantic web Technologies'. Dicho logo se puede ver en la figura 4.47.

Figura 4.47: Logo de la aplicación

5. TESTEO DE CASOS

Como la finalidad de esta aplicación es ofrecer ayuda a los alumnos, resulta muy importante poder evaluar los resultados. Debe garantizarse que el feedback obtenido ofrece conocimiento veraz y útil capaz de ayudar al alumno en su proceso de aprendizaje. Los resultados se pueden evaluar mediante profesores o expertos que conozcan muy a fondo tanto el dominio de conocimiento como la forma de trabajo, problemas y errores a los que se enfrenta un alumno a la hora de aprender un lenguaje de programación.

Para poder dejar constancia de la respuesta de esta aplicación y preparar una futura evaluación de los resultados, se elaboraron una serie de casos de prueba sobre causas comunes que provocan errores en las compilaciones de los alumnos.

Estos casos de prueba consisten en compilar un código concreto que simule uno o varios errores de compilación típicos entre alumnos que están aprendiendo a programar, cuyas causas están identificadas. Para realizar la simulación de errores, se han utilizado tanto casos reales de alumnos (código erróneo creado por alumnos), como casos provocados manualmente para obtener el error deseado.

Se optó por simular causas complejas, para las que la aplicación necesita interactuar en varias ocasiones con el alumno hasta llegar a un resultado, evitando casos sencillos que devuelven una respuesta de forma inmediata o en sólo un paso.

Tras la compilación del código con errores y simular la interacción que tendría el alumno con la aplicación, se muestra el feedback con el conocimiento y enlaces que se ofrecen al alumno. De esta forma se puede juzgar si la respuesta de la aplicación tiene relación con la verdadera causa o causas de error.

Algunos de estos casos de pruebas, se reflejan a continuación. Para cada uno de ellos se indica:

- Definición del error: explicación de la causa real que ha producido el error de compilación indicado.
- Código que produce el caso de prueba.
- Respuesta de la compilación: texto que aparece en la línea de comandos identificando los errores de compilación ocurridos. Otra forma de comprobar si el feedback generado es útil, es comparar este texto con la especificación de la causa del error que ofrece la aplicación. Como se puede comprobar, esta última es mucho más explicativa y específica, y además incluye una propuesta de solución.

- Código que produce el caso de prueba. Ejemplo de código que se ha creado o identificado entre casos reales de alumnos, para simular el error deseado. Las líneas de código que provocan el error, se visualizan en rojo.
- Respuesta de la aplicación: Esta respuesta se compone de una explicación de la causa del error, una propuesta de solución y una serie de enlaces a recursos web de la asignatura que ofrecen conocimiento sobre la causa específica su error. Los enlaces del tipo ***sentencias_control.html#error*** (sección X)', se corresponden con los recursos web creados durante el desarrollo de este proyecto y que están almacenados en local. Entre paréntesis se especifica la sección concreta a la que se refiere el enlace.

5.1. Caso de prueba 1. Falta añadir el prefijo 'extern'

Definición del error

Error: `'nombre' no se declaró aquí (primer uso en esta función)`

Causado porque no se ha añadido el prefijo 'extern' a la variable 'nombre', cuya declaración se encuentra en otro fichero.

Código que produce el caso de prueba

Fichero main.c

```
#include "lista.h"
#include "datos.h"

int main(int argc, char **argv){
    char *s="HOLA1";
    inicializoLista(&listaMain);
    sacarFoto(&listaMain,s);
    s="HOLA2";
        .
        .
        .
}
```

Fichero lista.c

```
#include "lista.h"

Lista listaMain;
```

Respuesta de la compilación

Al compilar con el siguiente comando:
`gcc -o proyecto *.c`

obtenemos la siguiente respuesta:

```
main.c: En la función 'main':
main.c:14:20: error: 'listaMain' no se declaró aquí
```

```
(primer uso en esta función)
main.c:14:20: nota: cada identificador sin declarar se
reporta sólo una vez para cada función que aparece
```

Respuesta de la aplicación

Causa: El error se ha producido porque al no especificar con el prefijo 'extern' que la variable listaMain esta declarada en otro fichero.c, el compilador no encuentra ninguna declaración.

Solución: Añada el prefijo 'extern' a la variable listaMain antes de usarla.

Enlaces a teoría y ejercicios relacionados con el error:

Teoría:

- Enlaces aconsejados para consultar y comprender las causas del error:
http://www.it.uc3m.es/labas/course_notes/variables_es.html#variables_static
http://www.it.uc3m.es/labas/course_notes/program_structure_es.html#c_program

- Enlaces a conceptos y elementos que aparecen en el error:
http://www.it.uc3m.es/labas/course_notes/dynamic_memory_stack_es.html
http://www.it.uc3m.es/labas/course_notes/variables_es.html#variables_shadowing
http://www.it.uc3m.es/labas/course_notes/variables_es.html#variables_scope

Ejercicios:

http://www.it.uc3m.es/labas/course_notes/program_structure_activities_es.html#program_structure_activities_program_elements
http://www.it.uc3m.es/labas/course_notes/program_structure_es.html
http://www.it.uc3m.es/labas/course_notes/variables_exercises_es.html

5.2. Caso de prueba 2. Falta incluir un fichero.h

Definición del error

Error: `'nombre' no se declaró aquí (primer uso en esta función)`

Causado porque no se ha incluido el 'fichero.h' donde se declara la variable 'nombre'.

Código que produce el caso de prueba

Fichero lista.h

```
Lista listaMain;
```

Fichero main.c

```
#include "datos.h"

int main(int argc, char **argv){
    char *s="HOLA1";
    inicializoLista(&listaMain);
```

```
    sacarFoto(&listaMain,s);
    s="HOLA2";
        .
        .
        .
}
```

Respuesta de la compilación

Al compilar con el siguiente comando:
`gcc -o proyecto *.c`

obtenemos la siguiente respuesta:

```
main.c: En la función 'main':
main.c:14:20: error: 'listaMain' no se declaró aquí
(primer uso en esta función)
main.c:14:20: nota: cada identificador sin declarar se
reporta sólo una vez para cada función que aparece
```

Respuesta de la aplicación

Causa: El error se ha producido porque el preprocesador no ha añadido el fichero.h a su código y por lo tanto no se encuentra la declaración de la variable lista.

Solución: Añada mediante la sentencia include el fichero.h y asegúrese de que la ruta y el nombre del mismo son correctos.

Enlaces a teoría y ejercicios relacionados con el error:

Teoría:

- Enlaces aconsejados para consultar y comprender las causas del error:
http://www.it.uc3m.es/labas/course_notes/program_structure_es.html#c_program
http://www.it.uc3m.es/labas/course_notes/variables_es.html#variables_static
http://www.it.uc3m.es/labas/course_notes/dynamic_memory_stack_es.html
http://www.it.uc3m.es/labas/course_notes/variables_es.html#variables_shadowing
http://www.it.uc3m.es/labas/course_notes/variables_es.html#variables_scope

Ejercicios:

http://www.it.uc3m.es/labas/course_notes/program_structure_es.html
http://www.it.uc3m.es/labas/course_notes/program_structure_activities_es.html#program_structure_activities_program_elements
http://www.it.uc3m.es/labas/course_notes/variables_exercises_es.html

5.3. Caso de prueba 3. Número de argumentos incorrecto

Definición del error

Error: `tipos en conflicto para 'nombre_funcion'`

Causado porque el número de argumentos declarados en el prototipo de la función 'nombre_funcion' no coincide con el de la definición de dicha función.

Código que produce el caso de prueba

Fichero lista.h

```
int inserto_Comentario(Lista lista, int id, char *comment);
```

Fichero lista.c

```
#include "lista.h"

int inserto_Comentario(Lista lista, char *comment){
    Nodo *aux;
    int long_antigua=-1;
    FILE *f;
        .
        .
        .
    return 0;
}
```

Respuesta de la compilación

Al compilar con el siguiente comando:

```
gcc -o proyecto *.c -lm
```

obtenemos la siguiente respuesta:

```
lista.c:150:5: error: tipos en conflictos para
'inserto_Comentario'
lista.h:42:5: nota: la declaración precia de
'inserto_Comentario' estaba aquí
```

Respuesta de la aplicación

Causa: El error se ha producido porque el número de argumentos declarados en el prototipo de la función inserto_Comentario no coincide con el de su definición.

Solución: Debe poner el mismo número de argumentos tanto en el prototipo como en la definición de la función inserto_Comentario.
Puede cambiar el nombre de los argumentos, pero no el tipo o el número.

Enlaces a teoría y ejercicios relacionados con el error:

Teoría:

- Enlaces aconsejados para consultar y comprender las causas del error:
http://www.it.uc3m.es/labas/course_notes/functions_es.html
http://www.it.uc3m.es/labas/course_notes/functions_prototypes_es.html
http://www.it.uc3m.es/labas/course_notes/functions_passing_parameters_es.html
http://www.it.uc3m.es/labas/course_notes/functions_es.html#functions_definitions

- Enlaces a conceptos y elementos que aparecen en el error:
http://www.it.uc3m.es/labas/course_notes/program_structure_es.html#c_program
http://www.it.uc3m.es/labas/course_notes/functions_passing_parameters_es.html#functions_tables_as_parameters

- Enlaces a conceptos generales:
http://www.it.uc3m.es/labas/course_notes/functions_static_es.html

Ejercicios:

http://www.it.uc3m.es/labas/course_notes/function_selfassessment_es.html
http://www.it.uc3m.es/labas/course_notes/program_structure_activities_es.html#program_structure_activities_program_elements
http://www.it.uc3m.es/labas/course_notes/program_structure_es.html
http://www.it.uc3m.es/labas/course_notes/functions_exercises_es.html

5.4. Caso de prueba 4. Demasiados tipos de datos definidos

Definición del error

Error: `dos o más tipos de datos en los especificadores de la declaración`

Causado porque se le ha asignado a la función, bien en su prototipo o en su definición, varios tipos de datos.

Código que produce el caso de prueba

Fichero array_of_pointers.c

```
#include <stdio.h>
#define NUM 7

int void print_strings(char **str1);

int main(void)
{
    char *str[NUM] =
    {"Sunday","Monday","Tuesday","Wednesday","Thursday",
    "Friday", "Saturday"};
    print_strings(str);
    return 0;
}

int void print_strings(char **str1)
{
    int i;
    for (i = 0;i<NUM;i++)
    {
        printf("%s\n",str1[i]);
    }
```

```
    return 0;
}
```

Respuesta de la compilación

Al compilar con el siguiente comando:
`gcc -o array_of_pointers array_of_pointers.c`

obtenemos la siguiente respuesta:

```
array_of_pointers.c:4:5: error: dos o más tipos de datos en
los especificadores de la declaración
array_of_pointers.c:14:5: error: dos o más tipos de datos en
los especificadores de la declaración
```

Respuesta de la aplicación

Causa: El error se ha producido porque no se puede asignar a lo que devuelve una función, más de un tipo de datos.

Solución: Corrige el prototipo o la definición de la función y pon un único tipo de datos.

Enlaces a teoría y ejercicios relacionados con el error:

Teoría:

- Enlaces aconsejados para consultar y comprender las causas del error:
http://www.it.uc3m.es/labas/course_notes/functions_es.html
http://www.it.uc3m.es/labas/course_notes/functions_prototypes_es.html

- Enlaces a conceptos y elementos que aparecen en el error:
http://www.it.uc3m.es/labas/course_notes/data_types_es.html
http://www.it.uc3m.es/labas/course_notes/data_types_es.html#data_types_size_of_basic_data_types
http://www.it.uc3m.es/labas/course_notes/functions_passing_parameters_es.html
http://www.it.uc3m.es/labas/course_notes/functions_es.html#functions_definitions
http://www.it.uc3m.es/labas/course_notes/program_structure_es.html#c_program

- Enlaces a conceptos generales:
http://www.it.uc3m.es/labas/course_notes/functions_static_es.html

Ejercicios:

http://www.it.uc3m.es/labas/course_notes/function_selfassessment_es.html
http://www.it.uc3m.es/labas/course_notes/data_types_exercises_es.html
http://www.it.uc3m.es/labas/course_notes/program_structure_activities_es.html#program_structure_activities_program_elements
http://www.it.uc3m.es/labas/course_notes/program_structure_es.html
http://www.it.uc3m.es/labas/course_notes/functions_exercises_es.html

5.5. Caso de prueba 5. Demasiadas definiciones

Definición del error

Error: `redefinition de 'nombre_funcion'`

Causado porque existe una definición de la función 'nombre_funcion' en el fichero.c donde se produce el error, y otra en un fichero.h que se incluye en el primero mediante:
`#include "fichero.h"`

Código que produce el caso de prueba

Fichero enteros.h

```
int funcion_suma (int a, int b){
    return a + b;
}
```

Fichero enteros.c

```
#include 'enteros.h'

int funcion_suma (int a, int b){
    return a + b;
}
```

Respuesta de la compilación

Al compilar con el siguiente comando:
`gcc -o proyecto *.c`

obtenemos la siguiente respuesta:

```
enteros.c:3:5: error: redefinition de 'funcion_suma'
enteros.h:1:5: nota: la definición previa de 'funcion_suma'
estaba aquí
```

Respuesta de la aplicación

Causa: El error se ha producido porque el preprocesador ha añadido un fichero.h a tu código y por lo tanto hay más de una definición de la función función_suma.

Solución: Deja tan sólo una definición de la función donde te sea más útil.

Enlaces a teoría y ejercicios relacionados con el error:

Teoría:

- Enlaces aconsejados para consultar y comprender las causas del error:
http://www.it.uc3m.es/labas/course_notes/functions_es.html#functions_definition

- Enlaces a conceptos y elementos que aparecen en el error:
http://www.it.uc3m.es/labas/course_notes/functions_es.html
http://www.it.uc3m.es/labas/course_notes/functions_passing_parameters_es.html

http://www.it.uc3m.es/labas/course_notes/functions_es.html#functions_prototypes
http://www.it.uc3m.es/labas/course_notes/program_structure_es.html#c_program

- Enlaces a conceptos generales:
http://www.it.uc3m.es/labas/course_notes/functions_static_es.html

Ejercicios:

http://www.it.uc3m.es/labas/course_notes/function_selfassessment_es.html
http://www.it.uc3m.es/labas/course_notes/program_structure_es.html
http://www.it.uc3m.es/labas/course_notes/program_structure_activities_es.html#program_structure_activities_program_elements
http://www.it.uc3m.es/labas/course_notes/functions_exercises_es.html

5.6. Caso de prueba 6. Falta cerrar paréntesis

Definición del error

Error: `expected ')' before 'nombre'`

Causado porque al declarar un bucle do-while no se ha cerrado paréntesis ')' tras poner la condición del bucle.

Código que produce el caso de prueba

Fichero using_scanf_and_realloc_sol.c

```
int main()
{
    int num1;
    char *ptr_char = NULL;
    do
    {
        scanf("%d",&num1);
        if (num1!=0)
        {
            ptr_char = (char *)realloc(ptr_char,(num1+1)*
            sizeof(char));
            scanf("%s",ptr_char);
        }
    } while (num1 ;
    free(ptr_char);
    return 0;
}
```

Respuesta de la compilación

Al compilar con el siguiente comando:
`gcc using_scanf_and_realloc_sol.c -o scanf_and_realloc`

obtenemos la siguiente respuesta:

```
using_scanf_and_realloc_sol.c: En la función 'main':
using_scanf_and_realloc_sol.c:21:19: error: expected ')'
before ';' token
using_scanf_and_realloc_sol.c:24:1: error: expected ';'
before '}' token
using_scanf_and_realloc_sol.c:24:1: error: expected
declaration or statement at end of input
```

Respuesta de la aplicación

Causa: El error se ha producido porque no se ha encontrado un carácter de cerrar paréntesis,), después de abrirlo, (.

Solución: Añade un carácter,), donde corresponda.

Enlaces a teoría y ejercicios relacionados con el error:

Teoría:

- Enlaces aconsejados para consultar y comprender las causas del error:
sentencias_control.html#do-while (sección 2.2.3)

- Enlaces a conceptos generales:
sentencias_control.html#condicionales (sección 2.1)
sentencias_control.html#sentencias_de_control (sección 2)
sentencias_control.html#salto (sección 2.3)
sentencias_control.html#bucles (sección 2.2)

5.7. Caso de prueba 7. Referencia a un puntero incorrecta

Definición del error

Error: `argumento de tipo inválido '->'`

Causado porque la expresión que precede al operador '->' es el nombre de una variable de tipo estructura o unión y no está precedido del operador &.

Código que produce el caso de prueba

Fichero fichero.c

```
int guardar_datos_pacceso (nuestra_lista lista)
{
                .
                .
                .
    if(cadena==NULL)
    {
        guardado=0;
```

```
            return guardado;
        }
        else
        {
            auxessid = lista->inicio;
                .
                .
                .
            lista->tamanoLista-;
                .
                .
                .
        }
    }
```

Respuesta de la compilación

Al compilar con el siguiente comando:
`gcc -o proyecto *.c -lm`

obtenemos la siguiente respuesta:

```
ficheros.c: En la función 'guardar_datos_pacceso':
ficheros.c:630:21: error: argumento de tipo inválido '->'
(se tiene 'nuestra_lista')
ficheros.c:634:14: error: error: argumento de tipo inválido
('->' se tiene 'nuestra_lista')
ficheros.c:659:23: error: error: argumento de tipo inválido
('->' se tiene 'nuestra_lista')
ficheros.c:689:13: error: error: argumento de tipo inválido
('->' se tiene 'nuestra_lista')
```

Respuesta de la aplicación

Causa: El error se ha producido porque el operador '->' no va precedido de la expresión adecuada.

Solución: Para el acceso indirecto a un campo a través de una variable, se debe cumplir uno de los siguientes esquemas:

(&nombre_variable)->nombre_campo
nombre_variable.nombre_campo

Enlaces a teoría y ejercicios relacionados con el error:

Teoría:

- Enlaces aconsejados para consultar y comprender las causas del error:
http://www.it.uc3m.es/labas/course_notes/pointers_indirection_with_pointers_es.html#pointers_structure_indirection

http://www.it.uc3m.es/labas/course_notes/pointers_indirection_es.html

- Enlaces a conceptos y elementos que aparecen en el error:
http://www.it.uc3m.es/labas/course_notes/dynamic_memory_examples_es.html
http://www.it.uc3m.es/labas/course_notes/pointers_structures_as_parameters_es.html
http://www.it.uc3m.es/labas/course_notes/pointers_es.html#pointers_all_data_has_an_address
http://www.it.uc3m.es/labas/course_notes/pointers_assignment_es.html
http://www.it.uc3m.es/labas/course_notes/pointers_to_pointers_es.html
http://www.it.uc3m.es/labas/course_notes/pointers_indirection_use_es.html

- Enlaces a conceptos generales:
http://www.it.uc3m.es/labas/course_notes/pointers_data_type_es.html
http://www.it.uc3m.es/labas/course_notes/dynamic_memory_arrays_and_pointers_es.html
http://www.it.uc3m.es/labas/course_notes/pointers_es.html
http://www.it.uc3m.es/labas/course_notes/data_types_structured_data_types_es.html

Ejercicios:

http://www.it.uc3m.es/labas/course_notes/pointers_indirection_use_evaluation_es.html
http://www.it.uc3m.es/labas/course_notes/pointers_indirection_evaluation_es.html
http://www.it.uc3m.es/labas/course_notes/pointers_assignment_es.html#pointers_assignment_evaluation
http://www.it.uc3m.es/labas/course_notes/pointers_data_type_es.html#pointers_data_type_evaluation

5.8. Caso de prueba 8. Referencia a una estructura incorrecta

Definición del error

Error: `petición del miembro 'nombre' en algo que no es una estructura o unión`

Causado porque lo que precede al operador '.' es un puntero a una estructura o unión precedido del operador '*', pero ambas cosas no van entre paréntesis.

Código que produce el caso de prueba

Fichero studentList t.c

```
int mediaTotal(struct student *lista){
    int num_alumnos=numAlumnos(lista);
    int suma_alumno=0;
    int suma_parcial=0;
    if (num_alumnos == 0)
        return 0;
```

```
        struct notas *aux;
        while (lista !=NULL){
            if (lista->numNotas!=0){
                aux= *lista.scores_ini;
                suma_alumno=0;
                while (aux!=NULL){
                    suma_alumno=suma_alumno+aux->califico;
                    aux=aux->sig;
                }
                suma_parcial=suma_parcial+
                (suma_alumno/lista->numNotas);
            }
            lista = lista->sig;
        }
        return suma_parcial/num_alumnos;
    }
```

Respuesta de la compilación

Al compilar con el siguiente comando:

```
gcc -o proyecto *.c
```

obtenemos la siguiente respuesta:

```
studentList.c: En la función 'mediaTotal':
studentList.c:148:18: error: petición del miembro
'scores_ini' en algo que no es una estructura o unión
```

Respuesta de la aplicación

Causa: El error se ha producido porque el operador . no va precedido de la expresión adecuada.

Solución: Para el acceso indirecto al campo score_ini a través de un puntero, se debe cumplir uno de los siguientes esquemas:

nombre_puntero->score_ini
(*nombre_puntero).score_ini

Enlaces a teoría y ejercicios relacionados con el error:

Teoría:

- Enlaces aconsejados para consultar y comprender las causas del error:
http://www.it.uc3m.es/labas/course_notes/pointers_indirection_with_pointers_es.html#pointers_structure_indirection
http://www.it.uc3m.es/labas/course_notes/pointers_indirection_es.html

- Enlaces a conceptos y elementos que aparecen en el error:
http://www.it.uc3m.es/labas/course_notes/dynamic_memory_examples_es.html
http://www.it.uc3m.es/labas/course_notes/pointers_structures_as_parameters_es.html

http://www.it.uc3m.es/labas/course_notes/pointers_es.html#pointers_all_data_has_an_address
http://www.it.uc3m.es/labas/course_notes/pointers_assignment_es.html
http://www.it.uc3m.es/labas/course_notes/pointers_to_pointers_es.html
http://www.it.uc3m.es/labas/course_notes/pointers_indirection_use_es.html

- Enlaces a conceptos generales:
http://www.it.uc3m.es/labas/course_notes/pointers_data_type_es.html
http://www.it.uc3m.es/labas/course_notes/dynamic_memory_arrays_and_pointers_es.html
http://www.it.uc3m.es/labas/course_notes/pointers_es.html
http://www.it.uc3m.es/labas/course_notes/data_types_structured_data_types_es.html

Ejercicios:

http://www.it.uc3m.es/labas/course_notes/pointers_indirection_use_evaluation_es.html
http://www.it.uc3m.es/labas/course_notes/pointers_indirection_evaluation_es.html
http://www.it.uc3m.es/labas/course_notes/pointers_assignment_es.html#pointers_assignment_evaluation
http://www.it.uc3m.es/labas/course_notes/pointers_data_type_es.html#pointers_data_type_evaluation

5.9. Caso de prueba 9. Llamada a una función incorrecta

Definición del error

Error: `faltan argumentos para la función 'nombre_función'`

Causado porque el número de argumentos de la definición de la función 'nombre_función', no coincide con el de la llamada a dicha función.

Código que produce el caso de prueba

Fichero lista.h

```
int main(int argc, char **argv)
{
    struct coleccion c
                .
                .
                .
    switch(n)
    {
        case 1:
                .
                .
                .
        case 4:
            sacar_foto(&c);
```

```
            break;
                .
                .
                .
}
```

Fichero Coleccion_fotos.c

```
int sacar_foto(struct coleccion *c,struct tm *tm){
                .
                .
                .
    return 0;
}
```

Fichero Coleccion_fotos.h

```
int sacar_foto(struct coleccion *c,struct tm *tm)
```

Respuesta de la compilación

Al compilar con el siguiente comando:
`gcc -o proyecto *.c -lm`

obtenemos la siguiente respuesta:

```
main.c: En la función 'main':
main.c:191:3: error: faltan argumentos para la función
'sacar_foto'
Coleccion_fotos.h:54:5: nota: se declara aquí
```

Respuesta de la aplicación

Causa: El error se ha producido porque el número de argumentos definidos para la función sacar_foto no coincide con el número de parámetros que se le pasan a su llamada.

Solución: Debe poner en la llamada a la función sacar_foto el mismo número de argumentos que se declararon en su definición.
Puede cambiar el nombre de los argumentos, pero no el tipo o el número.

Enlaces a teoría y ejercicios relacionados con el error:

Teoría:

- Enlaces aconsejados para consultar y comprender las causas del error:
http://www.it.uc3m.es/labas/course_notes/functions_es.html#functions_definition
http://www.it.uc3m.es/labas/course_notes/functions_passing_parameters_es.html
http://www.it.uc3m.es/labas/course_notes/functions_es.html
http://www.it.uc3m.es/labas/course_notes/functions_es.html#functions_prototypes

- Enlaces a conceptos y elementos que aparecen en el error:
http://www.it.uc3m.es/labas/course_notes/functions_passing_parameters_es.html#functions_tables_as_parameters

http://www.it.uc3m.es/labas/course_notes/program_structure_es.html#c_program
http://www.it.uc3m.es/labas/course_notes/functions_static_es.html

- Enlaces a conceptos generales:
http://www.it.uc3m.es/labas/course_notes/variables_es.html#variables_static
http://www.it.uc3m.es/labas/course_notes/dynamic_memory_stack_es.html
http://www.it.uc3m.es/labas/course_notes/variables_es.html#variables_shadowing
http://www.it.uc3m.es/labas/course_notes/variables_es.html#variables_scope

Ejercicios:

http://www.it.uc3m.es/labas/course_notes/function_selfassessment_es.html
http://www.it.uc3m.es/labas/course_notes/program_structure_activities_es.html#program_structure_activities_program_elements
http://www.it.uc3m.es/labas/course_notes/functions_exercises_es.html
http://www.it.uc3m.es/labas/course_notes/program_structure_es.html

5.10. Caso de prueba 10. Declaración bucle for incorrecta

Definición del error

Error: `sólo se permiten las declaraciones iniciales del bucle 'for' en modo C99`

Causado porque la variable de control usada en el bucle for se declara dentro de la declaración del bucle.

Código que produce el caso de prueba

insertSortDichSearch.c

```
void print_array_integers(int *array, int size)
{
    printf("Array = ");
    for (int i=0; i<size; i++)
        printf("%i ",array[i]);
}
```

Respuesta de la compilación

Al compilar con el siguiente comando:
`gcc -o programa *.c -DSIZE=5`

obtenemos la siguiente respuesta:

```
insertSortDichSearch.c: En la función 'print_array_integers':
insertSortDichSearch.c:23:3: error: sólo se permiten las
declaraciones iniciales del bucle 'for' en modo C99
insertSortDichSearch.c:23:3: nota: use la optión -std=c99 o
-std=gnu99 para compilar su código
```

Respuesta de la aplicación

Causa: El error se ha producido porque la variable no se puede definir dentro del for.

Solución: Declara la variable antes de usarla en el bucle.

Enlaces a teoría y ejercicios relacionados con el error:

Teoría:

- Enlaces aconsejados para consultar y comprender las causas del error:
sentencias_control.html#error (sección 2.2.1.1)

- Enlaces a conceptos y elementos que aparecen en el error:
sentencias_control.html#for (sección 2.2.1)

- Enlaces a conceptos generales:
sentencias_control.html#condicionales (sección 2.1)
sentencias_control.html#sentencias_de_control (sección 2)
sentencias_control.html#salto (sección 2.3)
sentencias_control.html#bucles (sección 2.2)

5.11. Caso de prueba 11. Sentencia simple incorrecta

Definición del error

Error: `expected ';' before 'nombre'`

Causado porque hay una sentencia simple que no finaliza en ;.

Código que produce el caso de prueba

Fichero insertSortDichSearch.c

```
void insertion_sort(int *array, int size)
{
    int i, j, value;
    for (i=1; i <size; i++)
    {
                .
                .
                .
        for (j=i-1; ((j >= 0) && (array[j] >value)); j-)
        {
            array[j + 1] = array[j]
            printf( "array[%i+1] =array[%i]",j,j);
            print_array_integers(array,size);

                .
                .
                .
}
```

Respuesta de la compilación

Al compilar con el siguiente comando:
```
gcc -o programa *.c -DSIZE=5
```

obtenemos la siguiente respuesta:
```
insertSortDichSearch.c: En la función 'insertion_sort':
insertSortDichSearch.c:61:8: error: expected ';' before
'}' token
```

Respuesta de la aplicación

Causa: El error se ha producido porque no se encuentra el final de la sentencia y la sintaxis no es correcta.

Solución: Añade un ; al final de la sentencia simple para delimitarla y separarla de la siguiente.

Enlaces a teoría y ejercicios relacionados con el error:

Teoría:

- Enlaces aconsejados para consultar y comprender las causas del error:
sentencias_control.html#sentecias) (sección 2)

6. CONCLUSIONES Y LÍNEAS FUTURAS

6.1. Conclusiones

En este proyecto fin de carrera se han logrado cumplir los objetivos iniciales, habiéndose desarrollado una herramienta software que utilizando técnicas de Web Semántica y estableciendo un diálogo con los alumnos, es capaz de proporcionar un feedback y recomendaciones de recursos valiosos, de forma que les ayude a los alumnos a detectar las causas de algunos errores de compilación cuando programan en C. El feedback ofrece conocimiento que ayuda al alumno a comprender y corregir sus propios errores, y con ello facilitar el aprendizaje. Concretamente, la respuesta que genera la aplicación se compone de dos aspectos:

- Conocimiento sobre las causas que han provocado los errores de compilación y una propuesta de solución.
- Listado de enlaces a aquellas páginas web (de entre todas las que conforman el temario de la asignatura 'Arquitectura de Sistemas') que ofrecen conocimiento sobre la causa específica de los errores.

Para llevar a cabo este proyecto se han utilizado multitud de tecnologías y conocimientos tales como manejo de bases de datos, comunicación entre sistemas y programa, interoperatibilidad entre los datos, desarrollo de algoritmos de búsqueda, utilización de Protegé para modelado de ontologías, utilización del entrono de Web semántica Jena o desarrollo de una interfaz gráfica para comunicación y presentación al usuario.

Además se ha llevado a cabo un trabajo de recopilación del estado del arte sobre la Web Semántica y su aplicación en educación. También se ha debido realizar una revisión del propio lenguaje de programación en C y de ciertos errores de compilación, para poder elaborar ontologías con contenido didáctico.

Como la aplicación desarrollada es una herramienta de ayuda, la idea es que el alumno pueda decidir si ejecutarla o no. Dicha ejecución se realiza de forma cómoda y sencilla con tan sólo poner un comando en la línea de comandos. Hasta que la aplicación no es capaz de ofrecer algún tipo de ayuda o información, se ejecuta de forma pasiva para no inferir en el trabajo del alumno.

Aunque se han logrado alcanzar los objetivos, existen algunos problemas identificados durante el desarrollo de este proyecto que se pueden abordar en un futuro. Un ejemplo de

ellos radica en el entorno de integración de la aplicación. Aunque se ha logrado instalar en la máquina virtual deseada, su integración a otros sistemas operativos es compleja. Tampoco se debe olvidar que, para detectar los errores de compilación ocurridos, se parte de una herramienta ya existente en dicha máquina virtual, por lo que habría que elaborar algo similar para su uso en otros entornos.

Además de cumplir con los objetivos iniciales, se ha completado la funcionalidad de la aplicación realizando dos tareas nuevas, fruto de las necesidades encontradas durante el desarrollo:

- Se detectó una debilidad producida por la existencia de ciertas causas de errores que no se pueden asociar a ningún recurso web de los que conforman la documentación de la asignatura, pues no hay conocimiento sobre ellas. Un ejemplo de esta situación son todas las causas relacionadas con una errónea definición de bucles, pues esta información no se contempla en la asignatura. Como solución a este problema, se ha optado por generar nuevos recursos que cubran las causa de errores que no se pueden apoyar en ningún de los recurso existente.

- La otra funcionalidad añadida a posteriori, es la opción de buscar recursos que traten sobre uno o varios conceptos específicos. De esta forma el alumno puede usar la aplicación como buscador seleccionando el concepto o conceptos de los que quiere consultar información y ahorrándose una búsqueda manual entre los apuntes de la asignatura.

6.2. Líneas futuras

Actualmente se está utilizando la aplicación desarrollada en una experiencia real en la asignatura 'Arquitectura de Sistemas', por lo que de cara a acciones futuras se pretende realizar evaluaciones por parte de alumnos y profesores.

Por un lado los alumnos probarán y juzgaran si realmente el feedback con conocimiento ofrecido es útil para su aprendizaje y les ha ayudado a solucionar sus errores. Para realizar una evaluación más profunda, sería interesante que los alumnos rellenaran una encuesta respondiendo a ciertas preguntas como: ¿la consulta de tu código para poder contestar a ciertas preguntas de la aplicación te ha ayudado a identificar problemas o conceptos?, ¿las preguntas que te ha formulado la aplicación sobre tu código han sido correctas y fáciles de entender?, ¿el lenguaje técnico usado en la aplicación se adecua a tu nivel de conocimientos?, ¿la interfaz gráfica te resulta agradable y fácil de usar?, ¿los mensajes explicativos que se van mostrando en el avance de la interacción te han resultado útiles y fáciles de entender?, o ¿te ha resultado sencilla la interacción con la aplicación?.

Por otro lado, la aplicación deberá ser juzgada por profesores con habilidades para juzgar si el conocimiento y los enlaces devueltos son correctos y están realmente relacionados con las causas de los errores de los alumnos.

La tecnología desarrollada puede servir como base para seguir desarrollando técnicas de servicios de personalización de contenidos para eliminar la sobrecarga de información que la red actual genera en algunas ocasiones. Además, partiendo de este proyecto se pueden

realizar otros similares pero orientados a otros dominios de conocimiento, por ejemplo para ayudar en el aprendizaje de otros lenguajes de programación como JAVA.

Las numerosas ontologías que se han desarrollado podrán ser reutilizadas por otras aplicaciones o herramientas, o como base para crear nuevas ontologías más complejas.

Como mejoras más importantes a realizar se han identificado las siguientes:

- Tratar todos los errores producidos en una misma compilación en conjunto, ya que suelen estar fuertemente relacionados y unos influyen sobre otros. En el proyecto actual cada error se trata de forma individual, sin tener en cuenta el resto.
- Extensión a más errores de compilación que puedan cometer los alumnos, pues en el presente proyecto sólo se ha trabajado con 12 (aunque todos ellos se encuentran entre los errores de compilación más comunes).
- Generar retroalimentación adaptativa basado en interacciones anteriores que construyan un perfil del alumno.
- Incluir una herramienta que sea capaz de buscar e interpretar en el propio código donde se ha producido el error, ciertos elementos como declaraciones de variables, tipo de dato que devuelve una función, o tipo y número de parámetros de una función. De esta forma se evitarían algunas de las preguntas que se le plantean al alumno y la identificación de la causa que ha provocado el error se realizaría en menos interacciones.
- Mejorar las ontologías desarrolladas añadiendo nuevos conceptos e instancias de clase.
- Desarrollo de razonadores sobre ontologías en OWL más óptimos.
- Mejorar los tiempos de respuesta de la aplicación. A pesar de que los obtenidos en este proyecto son óptimos y lo suficientemente cortos como para garantizar una adecuada interacción con el usuario, el tiempo de respuesta no se percibe como inmediato.
- Disminuir los recursos, sin olvidar que en muchas ocasiones la disminución de recursos conlleva un aumento del tiempo de reacción de la aplicación. Por lo que existe un compromiso entre tiempo de respuesta y recursos consumidos que hay que definir.
- Adaptación y generalización a otros entornos como Windows o Mac.

A. ANEXO: Manual de instalación y mantenimiento

En este anexo se especifican los pasos a seguir para la instalación y ejecución de la aplicación final. También se incluye un manual que describe el mantenimiento necesario para mantenerla actualizada ante posibles cambios de los recursos web de la asignatura a la que da soporte.

A.1. Instalación

En primer lugar debemos disponer de la Máquina Virtual de la asignatura 'Arquitectura de Sistemas' y asegurarnos de que tiene integrada la herramienta 'gcc' capaz de escribir en un fichero todos los errores de compilación producidos por el alumno dentro de dicho entorno.

La siguiente enumeración muestra una lista de todos los programas y herramientas que son necesarios para poder ejecutar la aplicación. Deberán formar parte de nuestro entorno, por lo que tienen que ser descargados e instalados en la Máquina Virtual.

Para el desarrollo de este proyecto se han utilizado las versiones indicadas, pero se pueden usar otras posteriores ya que en todos el apoyo a una versión implica soporte para las versiones posteriores (a menos que se indique lo contrario en una versión determinada). Todos ellos son libres y se pueden descargar desde los enlaces indicados:

1. **Java JDK 1.6.0.27**: Entorno de desarrollo para construir aplicaciones, applets y componentes utilizando el lenguaje de programación Java. Su descarga está disponible en el siguiente enlace:

   http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-javase6-419409.html

2. **Jena 2.6.4**: Proporciona un conjunto de herramientas y bibliotecas de Java para desarrollar aplicaciones de Web Semántica. Su descarga está disponible en el siguiente enlace:

   http://www.apache.org/dist/jena

3. **SDB 1.3.4**: Componente de Jena para el almacenamiento de RDF y consultas SPARQL. Su descarga está disponible en el siguiente enlace:

   http://www.apache.org/dist/jena

4. **MySQL 5.5.20-Obuntu0.12.04.2**: Una de las bases de datos compatibles con SDB, y la elegida para el desarrollo de este proyecto porque soporta Linux, es código libre, es fácil de configurar y tiene un buen rendimiento. Su descarga está disponible en el siguiente enlace:

   http://dev.mysql.com/downloads/mysql

5. **Conector MySQL para Java 5.1.19**: MySQL ofrece este controlador para el uso de bases de datos en plataformas Java. Su descarga está disponible en el siguiente enlace:

   http://dev.mysql.com/downloads/connector/j

Los ficheros de la aplicación se encuentran en una carpeta llamada 'tutor' que contiene todos los datos necesario para poder ejecutarla: código, imágenes, ontologías, html, bibliotecas, etc. La ruta de esta carpeta dentro de la máquina Virtual debe ser: `$HOME/tutor`

Todos las rutas que se indican en esta sección, suponen que la carpeta 'tutor' se encuentra en la ruta indicada, si no fuera así habría que adaptarlas a la ubicación real.

Si disponemos de la carpeta mencionada, la descarga e instalación de Jena, SDB y el conector MySQL no son necesarias, pues se incluyen ya en las siguientes rutas:

- Jena: `$HOME/tutor/jena/jena-2.6.4`
- SDB: `$HOME/tutor/sdb`
- Conector MySQL: `$HOME/tutor/sdb/lib/mysql-connector-java-5.1.19-bin.jar`

Una vez disponemos del entorno, los programas y herramientas indicadas en la sección anterior y los ficheros de la aplicación, se puede comenzar con la instalación. Los pasos a seguir para completar la instalación son los siguientes:

1. **Crear los 'store description' que usará SDB:**

   Concretamente necesitaremos dos, uno llamado 'sdb.ttl' que usaremos como descripción de la configuración de la base de datos, y otro más detallado para lograr la comunicación entre la aplicación Java y la base de datos, al que llamaremos 'sdb2.ttl'.

   Estos dos ficheros se pueden ver en las figuras A.1 y A.2, respectivamente, y deberán guardarse en la siguiente ruta: `$HOME/tutor/sdb`

   Se debe prestar especial atención al nombre de usuario (sdbUser) y contraseña (sdbPassword) que se definen en estos ficheros, ya que deben coincidir con los de MySQL. Para que estos ficheros sean válidos el nombre de usuario para la base de datos deberá ser 'root' y su password 'admin'.

   Si disponemos de la carpeta 'tutor' con todos los datos de la aplicación podemos saltarnos este punto pues, como se puede comprobar, estos ficheros ya existen en la ruta adecuada.

```
@prefix sdb:     <http://jena.hpl.hp.com/2007/sdb#> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ja:      <http://jena.hpl.hp.com/2005/11/Assembler#> .

# MySQL - InnoDB

<#store> rdf:type sdb:Store ;
    sdb:layout     "layout2" ;        # Layout name
    sdb:connection <#conn> ;
    sdb:engine     "InnoDB" ;      # MySQL specific
.

<#conn> rdf:type sdb:SDBConnection ;
    sdb:sdbType       "MySQL" ;    # Needed for JDBC URL
    sdb:sdbHost       "localhost" ;    # Host machine name
    sdb:sdbName       "project" ;    # Database Name
    sdb:driver        "com.mysql.jdbc.Driver" ;
    .
```

Figura A.1: Fichero 'sdb.ttl'

```
@prefix sdb:     <http://jena.hpl.hp.com/2007/sdb#> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ja:      <http://jena.hpl.hp.com/2005/11/Assembler#> .

# MySQL - InnoDB

<#store> rdf:type sdb:Store ;
    sdb:layout     "layout2" ;        # layout name
    sdb:connection <#conn> ;
    sdb:engine     "InnoDB" ;      # MySQL specific
.

<#conn> rdf:type sdb:SDBConnection ;
    sdb:sdbType       "MySQL" ;    # Needed for JDBC URL
    sdb:sdbHost       "localhost" ;   # Host machine name
    sdb:sdbName       "project" ;    # Database Name
    sdb:sdbUser       "root" ;       # Database user
    sdb:sdbPassword   "admin" ;      # Database password.
    sdb:driver        "com.mysql.jdbc.Driver" ;
.
```

Figura A.2: Fichero 'sdb2.ttl'

2. **Definir variables de entornos necesarias para ejecutar los scripts de SDB:**

   Los scripts de SDB que ejecutaremos más adelante para cargar información en la base de datos, necesitan la definición de las siguientes variables de entorno:

   - SDBROOT: Define la ruta donde se encuentran los datos y bibliotecas de SBD.
   - SDB_USER: Indica el nombre de usuario de la base de datos. Debe coincidir con el indicado en los ficheros .ttl.

- SDB_PASSWORD: Define la contraseña del usuario de la base de datos indicado en SDB_USER. Debe coincidir con la incluida en los ficheros .ttl.
- SDB_JDBC: Indica la ruta donde se encuentra el conector MySQL para Java.

Para poder compilar y ejecutar más tarde la aplicación desde línea de comandos, debemos añadir dos variables de entorno más:

- PATH: Añadimos a esta variable la ruta donde se encuentra el ejecutable de Java para que sus herramientas estén disponibles desde línea de comandos.
- CLASSPATH: Con las rutas de todas las bibliotecas de SDB y Jena, junto a la ruta del conector MySQL para Java.

Si queremos que estas variables de entornos estén definidas siempre que usemos la línea de comandos de la máquina virtual, se deben añadir al final del fichero '.profile' que se encuentra en la ruta: `$HOME`. En la figura A.3 se puede ver como serían las definiciones de estas variables adaptadas a nuestro entorno.

Para poder compilar y ejecutar la aplicación sin errores, vamos a añadir la ruta de la carpeta 'tutor' con todos los ficheros de la aplicación al path de nuestro sistema. Esto se realiza mediante la siguiente sentencia:

```
export PATH=$PATH:$HOME/.tutor
```

Para que sea persistente, añadimos la sentencia anterior en el fichero '.bashrc' que se encuentra en la ruta: `$HOME`.

3. **Crear la base de datos en MySQL:**

   El nombre de la base de datos (sdbName) debe ser el mismo que el indicado en los archivos .ttl. Para que coincida con el valor definido anteriormente, debe llamarse 'project'.

   Si queremos crear una base de datos con este nombre tan sólo tenemos que conectarnos o entrar en nuestro servidor MySQL y poner el siguiente comando:

```
create database project
```

4. **Añadir datos a la base de datos:**

   En primer lugar debemos dar formato al store y configurar índices y tablas en la base de datos. Para ello nos situamos desde línea de comandos en la siguiente ruta: `$HOME/tutor/sdb`

   Y ejecutamos el siguiente comando:

```
bin/sdbconfig -sdb=sdb.ttl -create
```

```
export SDBROOT="$HOME/.tutor/sdb"

export SDB_USER="root"
export SDB_PASSWORD="admin"

export SDB_JDBC="$HOME/.tutor/sdb/lib/mysql-connector-java-
5.1.21-bin.jar"

export PATH="/usr/bin/java":$PATH

export CLASSPATH=$HOME/.tutor/jena/jena-2.6.4/lib/jena-
2.6.4.jar:$HOME/.tutor/jena/jena-2.6.4/lib/jena-2.6.4-
tests.jar:$HOME/.tutor/jena/jena-2.6.4/lib/iri-
0.8.jar:$HOME/.tutor/jena/jena-2.6.4/lib/arq-
2.8.7.jar:$HOME/.tutor/jena/jena-2.6.4/lib/lucene-core-
2.3.1.jar:$HOME/.tutor/jena/jena-2.6.4/lib/xercesImpl-
2.7.1.jar:$HOME/.tutor/jena/jena-2.6.4/lib/slf4j-api-
1.5.8.jar:$HOME/.tutor/jena/jena-2.6.4/lib/icu4j-
3.4.4.jar:$HOME/.tutor/jena/jena-2.6.4/lib/slf4j-log4j12-
1.5.8.jar:$HOME/.tutor/jena/jena-2.6.4/lib/junit-
4.5.jar:$HOME/.tutor/jena/jena-2.6.4/lib/log4j-
1.2.13.jar:$HOME/.tutor/jena/jena-2.6.4/lib/wstx-asl-
3.2.9.jar:$HOME/.tutor/jena/jena-2.6.4/lib/stax-api-
1.0.1.jar:$HOME/.tutor/jena/jena-2.6.4/lib-src/iri-08-
sources:$HOME/.tutor/jena/jena-2.6.4/lib-src/jena-2.6.4-
sources:$HOME/.tutor/sdb/lib/arq-
2.8.7.jar:$HOME/.tutor/sdb/lib/arq-2.8.7-
tests.jar:$HOME/.tutor/sdb/lib/hsqldb-
1.8.0.10.jar:$HOME/.tutor/sdb/lib/icu4j-
3.4.4.jar:$HOME/.tutor/sdb/lib/iri-
0.8.jar:$HOME/.tutor/sdb/lib/jena-
2.6.4.jar:$HOME/.tutor/sdb/lib/jena-2.6.4-
tests.jar:$HOME/.tutor/sdb/lib/junit-
4.5.jar:$HOME/.tutor/sdb/lib/log4j-
1.2.13.jar:$HOME/.tutor/sdb/lib/lucene-core-
2.3.1:$HOME/.tutor/sdb/lib/mysql-connector-java-5.1.19-
bin.jar:$HOME/.tutor/sdb/lib/sdb-
1.3.4.jar:$HOME/.tutor/sdb/lib/sdb-1.3.4-
tests.jar:$HOME/.tutor/sdb/lib/slf4j-api-
1.5.8.jar:$HOME/.tutor/sdb/lib/slf4j-log4j12-
1.5.8.jar:$HOME/.tutor/sdb/lib/stax-api-
1.0.1.jar:$HOME/.tutor/sdb/lib/wstx-asl-
3.2.9.jar:$HOME/.tutor/sdb/lib/xercesImpl-2.7.1.jar
```

Figura A.3: Definición de variables a añadir en el fichero '.profile'

Desde el servidor MySQL se puede comprobar que efectivamente se han creado ciertas tablas en nuestra base de datos (Nodes, Prefixes, Quads y Triples). Para ello tan sólo debemos poner los siguientes comandos:

```
use project
show tables;
```

Los datos RDF que debemos cargar en la base de datos se componen de todas las ontologías (.owl) y un archivo llamado 'Etiquetado.rdf' que contiene el etiquetado de las páginas web con las que trabajamos, todo ello se encuentra en la carpeta:

```
$HOME/tutor/sdb/ontologias
```

Para añadirlos a la base de datos se usa el script 'sdbload' de SDB, por lo que debemos situarnos en la ruta: `$HOME/tutor/sdb`, e ir ejecutando un sdbload por cada archivo que deseamos cargar:

```
bin/sdbload -sdb=sdb.ttl ontologias/Etiquetado.rdf
bin/sdbload -sdb=sdb.ttl ontologias/C99ModeError.owl
bin/sdbload -sdb=sdb.ttl ontologias/
ConflictingTypesError.owl
bin/sdbload -sdb=sdb.ttl ontologias/
DeferencingPointerError.owl
bin/sdbload -sdb=sdb.ttl ontologias/
ExpectedCerrarParentesisError.owl
bin/sdbload -sdb=sdb.ttl ontologias/
ExpectedPuntoyComaError.owl
bin/sdbload -sdb=sdb.ttl ontologias/FewArgumentError.owl
bin/sdbload -sdb=sdb.ttl ontologias/
InvalidOperadorFlecha.owl
bin/sdbload -sdb=sdb.ttl ontologias/OntologiaC.owl
bin/sdbload -sdb=sdb.ttl ontologias/
RedefinitionFunctionError.owl
bin/sdbload -sdb=sdb.ttl ontologias/
RequestMemberError.owl
bin/sdbload -sdb=sdb.ttl ontologias/SeveralDatatypes.owl
bin/sdbload -sdb=sdb.ttl ontologias/
StatementBeforeError.owl
bin/sdbload -sdb=sdb.ttl ontologias/
VariableUndeclaredError.owl
```

5. **Compilar la aplicación:**

   Para poder compilar la aplicación se deben indicar todas las bibliotecas de SDB y Jena necesarias, así cómo la ruta del conector MySQL para Java. El comando necesario para compilar desde línea de comandos se muestra en la figura A.4.

   Si disponemos de la carpeta 'tutor' con todos los datos de la aplicación podemos saltarnos este punto pues, como se puede comprobar, ya están creados todos los ficheros '.class'.

```
javac -classpath $HOME/.tutor/jena/jena-2.6.4/lib/jena-
2.6.4.jar:$HOME/.tutor/jena/jena-2.6.4/lib/jena-2.6.4-
tests.jar:$HOME/.tutor/jena/jena-2.6.4/lib/iri-
0.8.jar:$HOME/.tutor/jena/jena-2.6.4/lib/arq-
2.8.7.jar:$HOME/.tutor/jena/jena-2.6.4/lib/lucene-core-
2.3.1.jar:$HOME/.tutor/jena/jena-2.6.4/lib/xercesImpl-
2.7.1.jar:$HOME/.tutor/jena/jena-2.6.4/lib/slf4j-api-
1.5.8.jar:$HOME/.tutor/jena/jena-2.6.4/lib/icu4j-
3.4.4.jar:$HOME/.tutor/jena/jena-2.6.4/lib/slf4j-log4j12-
1.5.8.jar:$HOME/.tutor/jena/jena-2.6.4/lib/junit-
4.5.jar:$HOME/.tutor/jena/jena-2.6.4/lib/log4j-
1.2.13.jar:$HOME/.tutor/jena/jena-2.6.4/lib/wstx-asl-
3.2.9.jar:$HOME/.tutor/jena/jena-2.6.4/lib/stax-api-
1.0.1.jar:$HOME/.tutor/jena/jena-2.6.4/lib-src/iri-08-
sources:$HOME/.tutor/jena/jena-2.6.4/lib-src/jena-2.6.4-
sources:$HOME/.tutor/sdb/lib/arq-
2.8.7.jar:$HOME/.tutor/sdb/lib/arq-2.8.7-
tests.jar:$HOME/.tutor/sdb/lib/hsqldb-
1.8.0.10.jar:$HOME/.tutor/sdb/lib/icu4j-
3.4.4.jar:$HOME/.tutor/sdb/lib/iri-
0.8.jar:$HOME/.tutor/sdb/lib/jena-
2.6.4.jar:$HOME/.tutor/sdb/lib/jena-2.6.4-
tests.jar:$HOME/.tutor/sdb/lib/junit-
4.5.jar:$HOME/.tutor/sdb/lib/log4j-
1.2.13.jar:$HOME/.tutor/sdb/lib/lucene-core-
2.3.1:$HOME/.tutor/sdb/lib/mysql-connector-java-5.1.19-
bin.jar:$HOME/.tutor/sdb/lib/sdb-
1.3.4.jar:$HOME/.tutor/sdb/lib/sdb-1.3.4-
tests.jar:$HOME/.tutor/sdb/lib/slf4j-api-
1.5.8.jar:$HOME/.tutor/sdb/lib/slf4j-log4j12-
1.5.8.jar:$HOME/.tutor/sdb/lib/stax-api-
1.0.1.jar:$HOME/.tutor/sdb/lib/wstx-asl-
3.2.9.jar:$HOME/.tutor/sdb/lib/xercesImpl-
2.7.1.jar:$HOME/.tutor/sdb/lib/mysql-connector-java-5.1.19-
bin.jar $HOME/.tutor/src/proyecto/*.java
```

Figura A.4: Comando de compilación de la aplicación

6. **Crear un script que ejecute la aplicación:**

   Para que la ejecución de nuestro aplicación resulte rápida y sencilla para los alumnos, vamos a crear un script para que se pueda ejecutar con tan sólo poner en línea de comandos (y desde cualquier ruta): `tutor.sh`

Dicho script se crearía añadiendo el texto de la figura A.5 a un fichero llamado 'tutor.sh'.

```
#!/bin/bash

cd $HOME/.tutor/src

java -classpath $HOME/.tutor/jena/jena-2.6.4/lib/jena-
2.6.4.jar:$HOME/.tutor/jena/jena-2.6.4/lib/jena-2.6.4-
tests.jar:$HOME/.tutor/jena/jena-2.6.4/lib/iri-
0.8.jar:$HOME/.tutor/jena/jena-2.6.4/lib/arq-
2.8.7.jar:$HOME/.tutor/jena/jena-2.6.4/lib/lucene-core-
2.3.1.jar:$HOME/.tutor/jena/jena-2.6.4/lib/xercesImpl-
2.7.1.jar:$HOME/.tutor/jena/jena-2.6.4/lib/slf4j-api-
1.5.8.jar:$HOME/.tutor/jena/jena-2.6.4/lib/icu4j-
3.4.4.jar:$HOME/.tutor/jena/jena-2.6.4/lib/junit-
4.5.jar:$HOME/.tutor/jena/jena-2.6.4/lib/log4j-
1.2.13.jar:$HOME/.tutor/jena/jena-2.6.4/lib/wstx-asl-
3.2.9.jar:$HOME/.tutor/jena/jena-2.6.4/lib/stax-api-
1.0.1.jar:$HOME/.tutor/jena/jena-2.6.4/lib-src/iri-08-
sources:$HOME/.tutor/jena/jena-2.6.4/lib-src/jena-2.6.4-
sources:$HOME/.tutor/sdb/lib/arq-
2.8.7.jar:$HOME/.tutor/sdb/lib/arq-2.8.7-
tests.jar:$HOME/.tutor/sdb/lib/hsqldb-
1.8.0.10.jar:$HOME/.tutor/sdb/lib/icu4j-
3.4.4.jar:$HOME/.tutor/sdb/lib/iri-
0.8.jar:$HOME/.tutor/sdb/lib/jena-
2.6.4.jar:$HOME/.tutor/sdb/lib/jena-2.6.4-
tests.jar:$HOME/.tutor/sdb/lib/junit-
4.5.jar:$HOME/.tutor/sdb/lib/log4j-
1.2.13.jar:$HOME/.tutor/sdb/lib/lucene-core-
2.3.1:$HOME/.tutor/sdb/lib/mysql-connector-java-5.1.19-
bin.jar:$HOME/.tutor/sdb/lib/sdb-
1.3.4.jar:$HOME/.tutor/sdb/lib/sdb-1.3.4-
tests.jar:$HOME/.tutor/sdb/lib/slf4j-api-
1.5.8.jar:$HOME/.tutor/sdb/lib/slf4j-log4j12-
1.5.8.jar:$HOME/.tutor/sdb/lib/stax-api-
1.0.1.jar:$HOME/.tutor/sdb/lib/wstx-asl-
3.2.9.jar:$HOME/.tutor/sdb/lib/xercesImpl-
2.7.1.jar:$HOME/.tutor/sdb/lib/mysql-connector-java-5.1.19-
bin.jar:$HOME/.tutor/src proyecto.Inicio
```

Figura A.5: Fichero 'tutor.sh'

Si disponemos de la carpeta 'tutor' con todos los datos de la aplicación, la creación de este fichero no es necesaria pues ya se encuentra en la ruta: `$HOME/tutor`

Por último, se dan permisos de ejecución al script desde consola mediante la siguiente sentencia:

```
chmod +x tutor.sh
```

A.2. Ejecución de la aplicación

Tras una correcta instalación y dar permisos de ejecución al script que ejecuta la aplicación, ya estamos en disposición de ejecutar la aplicación de forma rápida y sencilla. Los alumnos que quieran usar esta aplicación de ayuda al aprendizaje, tan sólo tendrán que abrir la línea de comandos de la Máquina Virtual y, desde cualquier ruta, poner la siguiente sentencia: `tutor.sh`

Esto ejecuta la aplicación, pero esta no se hará visible hasta que el alumno compile y se genere información sobre la que ofrecer ayuda. De esta forma se minimiza el impacto que pueda tener la aplicación sobre el trabajo habitual del alumno.

La línea de comandos desde la que se ejecuta la aplicación debe permanecer abierta durante todo el tiempo en el que se desee disponer de la posible ayuda que la aplicación ofrece al alumno. Si se cerrase, la ejecución de la aplicación finalizaría.

Debido a ello, las compilaciones y demás acciones que se quieran desarrollar mientras mantenemos nuestra aplicación activa, se deben realizar en otra ventana de la línea de comandos de la Máquina Virtual.

A.3. Mantenimiento

Debido a que esta aplicación ofrece enlaces a recursos educativos que pueden cambiar tanto de contenido como de URL, se necesita un cierto mantenimiento que actualice los posibles cambios.

Concretamente hay que mantener actualizada la anotación de los recursos que se define en un archivo RDF llamado 'Etiquetado.rdf' (una explicación detallada de este archivo se puede consultar en la sección 4.2.) que se encuentra en la siguiente ruta:

`$HOME/tutor/sdb/ontologias`.

Las posibles causas por las que se deberían modificar las anotaciones RDF existentes son:

- Cambios en las URLs, bien porque se ha cambiado el nombre o secciones del recurso, o porque se ha modificado el servidor. En este caso habría que cambiar la URLs que se indican en el atributo 'rdf:about'.
- Cambios en el contenido del recurso. Habría que modificar los valores de la propiedad 'label' para adaptarlos al nuevo contenido.
- Eliminación de un recurso. En este supuesto habría que eliminar todas las etiquetas sobre ese recurso; es decir, borrar la tripleta 'Description' completa para que la aplicación nunca devuelva una referencia a un recurso que ya no existe.
- Creación de un recurso totalmente nuevos. En este caso habría que incluir una etiqueta 'Descripción' completa sobre el nuevo recurso, que incluya tanto su URL como la descripción de su contenido.

En el caso de que se haya modificado el servidor de los recursos, además se deben realizar una serie de cambios en código. Concretamente debemos dirigirnos a las líneas 573 y 575 del fichero 'Aplicacion.c' que se encuentra en la ruta: `$HOME/tutor/src/proyecto`, y cambiar el valor de la variable 'uri' que almacena el comienzo de las URLs donde se indica el servidor.

Bibliografía

[1] PEDRO J. MUÑOZ-MERINO, ABELARDO PARDO, MAREN SCHEFFEL, KATJA NIEMANN, MARTIN WOLPERS, DERICK LEONY, Y CARLOS DELGADO KLOOS, *An Ontological Framework for Adaptive Feedback to Support Students while Programming*, In International Semantic Web Conference 2011.

[2] BERNERS-LEE, T., HENDLER, J. Y LASSILA, O., *The Semantic Web*, Scientific American, 2001, 284 (5), 34-43.

[3] W3C WORLD WIDE WEB CONSORTIUM, http://www.w3.org

[4] M. A. ABIAN, *El futuro de la web: XML, RDF/RDFS, ontologias y la web semantica*, 2005.

[5] PABLO CASTELLS, *La web semántica*,
http://arantxa.ii.uam.es/častells/publications/castells-uclm03.pdf

[6] JOURNAL OF WEB SEMANTICS, http://www.elsevier.com/locate/websem

[7] THE SEMANTIC WEB - A NEW ETAI AREA, http://www.etaij.org/seweb

[8] ISWC: INTERNATIONAL SEMANTIC WEB CONFERENCE, http://iswc2013.semanticweb.org

[9] PAOLA CECILIA MIRANDA MUTIZÁBAL, *Pautas Para Implementar Servicios Basados En Web Semántica En Una Empresa de Telecomunicaciones Móviles*, 2009, http://cybertesis.uach.cl/tesis/uach/2009/bmfcim6721p/doc/bmfcim6721p.pdf

[10] O4E: ONTOLOGIES FOR EDUCATION, http://iiscs.wssu.edu/drupal/o4e_welcome

[11] DEVEDZIC, VLADAN, *Semantic Web and Education*, primera edición, Springer Verlag GmbH, 2006.

[12] AROYO, L. Y DICHEVA, D., *The New Challenges for E-learning: The Educational Semantic Web*, Educational Technology & Society - ETS, 2004, 7(4), 59-69.

[13] COORDINADO POR ANA LANDETA ETXEBERRÍA, *Buenas Prácticas de e-learning*, http://www.buenaspracticas-elearning.com

[14] N. HENZE AND W. NEJDL, *Logically characterizing adaptive educational hypermedia systems*, Proceedings of the joint international workshop on Adaptivity, personalizationand the semantic web, 2003.

[15] JELENA JOVANOVIC, DRAGAN GASEVIC Y VLADAN DEVEDZIC, *TANGRAM for Personalized Learning Using the Semantic Web Technologies*, Journal of emerging technologies in web intelligence, 1(1), Agosto 2009.

[16] NICOLA HENZE, PETER DOLOG, Y WOLFGANG NEJDL, *Reasoning and Ontologies for Personalized E-Learning in the Semantic Web*, Educational Technology & Society - ETS, 2004, 7(4), 82-97.

[17] JASON OHLER, *The Semantic Web in Education*, EDUCAUSE QUARTERLY, Number 4, 2008.

[18] JULIEN TANE, CHRISTOPH SCHMITZ Y GERD STUMME, *Courseware Watchdog*, http://cwatchdog.sourceforge.net

[19] DAML ONTOLOGY LIBRARY, http://www.daml.org/ontologies

[20] PROTEGE ONTOLOGY LIBRARY,

http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library

[21] WOP: THE WEB OF PATTERNS PROJECT, http://www-ist.massey.ac.nz/wop

[22] RDF, http://www.w3.org/RDF

[23] FRANCESCO RICCI, LIOR ROKACH Y BRACHA SHAPIRA, *Recommender Systems Handbook*, Springer, 2011.

[24] EATEL: EUROPEAN ASSOCIATION OF TECHNOLOGY ENHANCED LEARNING, http://ea-tel.eu

[25] KATRIEN VERBERT, NIKOS MANOUSELIS, XAVIER OCHOA, MARTIN WOLPERS, HENDRIK DRACHSLER, IVANA BOSNIC Y ERIK DUVAL, *Context-Aware Recommender Systems for Learning: A Survey and Future Challenges*, TLT, 2012, 5(4), 318-335.

[26] LEARNING OBJECT CONTEXT ONTOLOGIES - THE LOCO FRAMEWORK, http://jelenajovanovic.net/LOCO-Analyst/index.html

[27] XML, *Extensible Markup Language (XML)*, http://www.w3.org/XML

[28] DUBLIN CORE METADATA INITIATIVE, http://www.dublincore.org

[29] W3C WORLD WIDE WEB CONSORTIUM. (2007), *SPARQL Query Language for RDF*, http://www.w3.org/TR/rdf-sparql-query

[30] OWL, *Web-Ontology (WebOnt) Working Group*, http://www.w3.org/2001/sw/WebOnt

[31] APACHE JENA, http://jena.apache.org

[32] API JENA, http://jena.apache.org/documentation/javadoc/jena

[33] SDB, *Persistent triple stores using relational databases*, http://jena.apache.org/documentation/sdb

[34] PROTÉGÉ, http://protege.stanford.edu

[35] JOHAN HJELM, *Creating the Semantic Web with RDF: Professional Developer's Guide*, John Wiley and Sons, Inc. New York, NY, USA 2001.

[36] MATTHEW HORRIDGE, *A practical guide to building OWL ontologies using Protégé 4 and CO-ODE tools*, Edition 1.3, The University of Manchester, 2011.

[37] BOYCE, SINEAD Y PAHL, CLAUS, *Developing domain ontologies for course content*, Educational Technology & Society - ETS, 2007, 10(3), 275-288.

[38] JESÚS CONTRERAS Y J.A MARTÍNEZ COMECHE, *TUTORIAL ONTOLOGÍAS*, http://www.sedic.es/gt_normalizacion_tutorial_ontologias.pdf

[39] MAREN SCHEFFEL, KATJA NIEMANN, ABELARDO PARDO, DERICK LEONY, MARTIN FRIEDRICH, KERSTIN SCHMIDT, MARTIN WOLPERS Y CARLOS DELGADO KLOOS, *Usage Pattern Recognition in Student Activities*, EC-TEL, 2011, Volume 6964, 341-355.

[40] LILIA CHENITI-BELCADHI, NICOLA HENZE Y RAFIK BRAHAM, *An Assessment Framework for eLearning in the Semantic Web*, Distributed System Institut publication, Hannover, Germany, 2004, p.6.

[41] SAMPSON, D. G., LYTRAS, M. D., WAGNER, G., AND DIAZ, P, *Ontologies and the Semantic Web for E-learning*, Journal of Educational Technology and Society, 2004, 7(4), 26-28.

[42] NATALYA F. NOY AND DEBORAH L. MCGUINNESS, *Ontology Development 101: A Guide to Creating Your First Ontology*, http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html

[43] JENA, *SPARQL Tutorial*, http://jena.sourceforge.net/ARQ/Tutorial

[44] JENA, *An Introduction to RDF and the Jena RDF API*, http://jena.sourceforge.net/tutorial/RDF_API/index.

[45] I. HERMAN, *Introduction to the Semantic Web*, http://www.w3.org/2007/Talks/0423-Stavanger-IH

[46] W3C WORLD WIDE WEB CONSORTIUM, *Processing your data using N3 and Cwm*, http://www.w3.org/2000/10/swap/doc/Processing

[47] W3C WORLD WIDE WEB CONSORTIUM. (2008), *El W3C expone los datos en la Web con SPARQL*, http://www.w3c.es/Prensa/2008/nota080115_sparql